

HP Serviceguard A.11.20.00 for Linux の管理



ご注意

1. 本書に記載した内容は、予告なしに変更することがあります。
2. 本書は内容について細心の注意をもって作成いたしました。万が一ご不審な点や誤り、記載もれなど、お気づきの点がございましたら当社までお知らせください。
3. 当社は、お客様の誤った操作に起因する損害については、責任を負いかねますのでご了承ください。
4. 当社では、本書に関して特殊目的に対する適合性、市場性などについては、一切の保証をいたしかねます。また、備品、性能などに関連した損傷についても保証いたしかねます。
5. 当社提供外のソフトウェアの使用や信頼性についての責任は負いかねます。
6. 本書の内容の一部または全部を、無断でコピーしたり、他のプログラム言語に翻訳することは法律で禁止されています。
7. 本製品パッケージとして提供した本書や媒体は本製品用だけにお使いください。プログラムをコピーする場合はバックアップ用だけにしてください。プログラムをそのままの形で、あるいは変更を加えて第三者に販売することは固く禁じられています。

U.S. Government License

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

著作権

© Copyright 2001-2012 Hewlett-Packard Development Company, L.P.

本書には著作権によって保護されている内容が含まれています。本書の内容の一部または全部を著作者の許諾なしに複製、改変、および翻訳することは、著作権法下での許可事項を除き、禁止されています。

商標

HP Serviceguard® は、Hewlett-Packard Company の登録商標であり、著作権で保護されています。

NIST™ は、米国 Sun Microsystems, Inc.の商標です。

UNIX® は、The Open Group の登録商標です。

Linux® は、Linus Torvalds の登録商標です。

Red Hat® は、Red Hat Software, Inc.の登録商標です。

SUSE® は、SUSE AG、Novell Business の登録商標です。

納入後の保証について

- 保証の期間は、ご購入時に当社よりお出しした見積書に記載された期間とします。保証サービスは、当社の定める休日を除く月曜日から金曜日までの、午前 8 時 45 分から午後 5 時 30 分の範囲で無料で行います。当社で定めたシステム製品については出張修理を行い、その他の製品については当社にご返却いただいた上での引取り修理となります。当社が定める地域以外における出張修理対象製品の修理は、保証期間中においても技術者派遣費が有料となります。
- ソフトウェア製品の保証は上記にかかわらず、下記に定める範囲とさせていただきます。
 - ソフトウェア製品およびマニュアルは当社が供給した媒体物の破損、資料の落丁およびプログラムインストラクションが実行できない場合のみ保証いたします。
 - バグおよび前記以外の問題の解決は、別に締結するソフトウェアサポート契約に基づいて実施されます。
- 次のような場合には、保証期間内でも修理が有料となります。
 - 取扱説明書等に記載されている保証対象外部品の故障の場合。
 - 当社が供給していないソフトウェア、ハードウェア、または補用品の使用による故障の場合。
 - お客様の不適當または不十分な保守による故障の場合。
 - 当社が認めていない改造、酷使、誤使用または誤操作による故障の場合。
 - 納入後の移設が不適切であったための故障または損傷の場合。
 - 指定外の電源 (電圧、周波数) 使用または電源の異常による故障の場合。
 - 当社が定めた設置場所基準に適合しない場所での使用、および設置場所の不適當な保守による故障の場合。
 - 火災、地震、風水害、落雷、騒動、暴動、戦争行為、放射能汚染、およびその他天災地変等の不可抗力的事故による故障の場合。
- 当社で取り扱う製品は、ご需要先の特定目的に関する整合性の保証はいたしかねます。また、そこから生じる直接的、間接的損害に対しても責任を負いかねます。
- 当社で取り扱う製品を組み込みあるいは転売される場合は、最終需要先における直接的、間接的損害に対しては責任を負いかねます。
- 製品の保守、修理用部品の供給期間は、その製品の製造中止後 5 年間とさせていただきます。

本製品の修理については取扱説明書に記載されている最寄の事業所へお問い合わせください。

目次

出版履歴	15
原典	15
はじめに	16
1 Serviceguard for Linux の概要	18
Serviceguard for Linux について	18
フェイルオーバー	19
Serviceguard Manager の使用	20
Serviceguard Manager を使ったクラスターの監視	21
Serviceguard Manager を使ったクラスターの管理	21
Serviceguard Manager を使ったクラスターの構成	21
Serviceguard Manager の起動	21
構成の手順	21
2 Serviceguard for Linux のハードウェア構成について	23
冗長クラスター構成要素	23
冗長ネットワーク構成要素	23
規則と制限事項	24
冗長イーサネット構成	24
クロスサブネット構成	25
構成作業	25
制限事項	26
詳細情報	27
冗長ディスクストレージ	27
サポートされているディスクインターフェイス	27
ディスクモニター	27
ディスク構成例	28
冗長電源	28
3 Serviceguard のソフトウェア構成要素	29
Serviceguard のアーキテクチャー	29
Serviceguard のデーモン	29
構成デーモン: cmclconfd	30
クラスターデーモン: cmcld	30
ログデーモン: cmlogd	31
Network Manager デーモン: cmnetd	31
ロック LUN デーモン: cmdisklockd	31
サービスアシスタントデーモン: cmserviced	31
汎用リソースアシスタントデーモン: cmresourced	31
クォーラムサーバーデーモン: qs	31
ユーティリティデーモン: cmlockd	32
クラスター SNMP エージェントデーモン: cmsnmpd	32
クラスター WBEM エージェントデーモン: cmwbemd	32
プロキシデーモン: cmproxyd	32
クラスターマネージャーの動作	33
クラスターの構成	33
ハートビートメッセージ	33
クラスター全体の手動による起動	34
クラスターの自動起動	34
動的クラスター再編成	34
スプリットブレインの問題を回避するクラスター定足数	34
クラスターロック	35

クラスターロックとしてのロック LUN の使用.....	35
クラスターロックとしてのクォーラムサーバーの使用.....	36
クラスターロックなし	37
オンラインでクォーラム構成を変更したときの動作.....	37
パッケージマネージャーの動作.....	38
パッケージのタイプ.....	38
フェイルオーバーしないパッケージ.....	38
フェイルオーバーパッケージ.....	38
フェイルオーバーパッケージの構成	39
フェイルオーバーパッケージを実行/停止する時期と場所の決定	39
フェイルオーバーパッケージの切り替え動作.....	39
フェイルオーバーポリシー.....	41
自動交替用スタンバイ.....	42
フェイルバックポリシー.....	43
フェイルオーバー方針とフェイルバック方針の組み合わせ.....	45
汎用リソース監視サービスの使用.....	45
古いパッケージ構成ファイルの使用.....	46
パッケージの動作.....	47
パッケージの実行.....	47
制御スクリプトの起動前.....	48
実行スクリプトの実行中.....	49
実行スクリプトの正常終了および異常終了.....	50
cmrunserv を使ったサービスの起動	50
サービスの実行中.....	51
サービスやサブネットの異常終了時、または汎用リソースや依存関係が満たされない場合.....	51
コマンドによるパッケージの停止時.....	51
停止スクリプトの実行中.....	52
停止スクリプトの正常終了および異常終了.....	53
パッケージ制御スクリプトのエラー条件および終了条件.....	53
ネットワークマネージャーの動作	54
定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット.....	54
IP アドレスの種類.....	55
再配置可能 IP アドレスの追加と削除	55
負荷の分散	56
LAN インターフェイスのボンディング	56
負荷バランスを目的としたボンディング.....	58
LAN インターフェイスの監視と障害の検出: リンクレベル.....	59
LAN インターフェイスの監視と障害の検出: IP レベル.....	59
IP モニターを使用する理由.....	59
IP モニターの動作.....	60
障害および回復の検出時間.....	61
制約および制限事項.....	62
リンクレベルおよび IP レベルの障害の報告.....	62
パッケージ切り替えと再配置可能 IP アドレス.....	62
同じサブネット上での切り替え後のアドレス解決メッセージ	63
VLAN 構成.....	63
VLAN について.....	63
Linux VLAN のサポート	63
構成上の制限事項.....	64
その他のハートビート要件.....	64
データストレージ用のボリュームマネージャー.....	64
アレイ上のストレージ.....	65
ディスクの監視.....	65
LVM についての詳細.....	65
Persistent Reservation について.....	65

規則と制限事項.....	66
Persistent Reservation の動作.....	67
障害への応答	68
ノード障害発生時のリブート	68
ノードがタイムアウトしたときの動作.....	68
例	68
ハードウェア障害への応答	69
パッケージ障害とサービス障害への応答	70
パッケージ障害と汎用リソース障害への応答.....	70
サービスの再起動	71
ネットワーク通信障害	71
4 HA クラスターのプランニングと文書化	72
プランニング全般	72
Serviceguard のメモリ要件.....	72
拡張のプランニング	72
Serviceguard と VMware 仮想マシンの併用.....	73
クラスター構成の選択肢.....	73
ハードウェアのプランニング	74
SPU 情報	74
LAN 情報	74
共有ストレージ.....	75
FibreChannel.....	75
ストレージのマルチパス	75
ディスク入出力情報	75
ハードウェア構成用ワークシート	76
電源のプランニング	76
電源構成用ワークシート	77
クラスターロックのプランニング.....	77
クラスターロックの要件.....	77
拡張のプランニング.....	77
クォーラムサーバーの使用.....	78
クォーラムサーバー用ワークシート	78
ボリュームマネージャーのプランニング	78
ボリュームグループと物理ボリュームのワークシート.....	78
クラスター構成のプランニング	79
ハートビートサブネットとクラスターの再編成時間	79
ホスト名アドレスファミリーについて: IPv4 専用、IPv6 専用、および混合モード.....	79
IPv4 専用モードについて.....	80
IPv6 専用モードについて.....	80
IPv6 専用モードの規則と制限事項.....	80
IPv6 専用モードの推奨事項.....	81
混合モードについて.....	82
混合モードの規則と制限事項.....	82
クラスター構成のパラメーター	82
クラスター構成: 次の手順	92
パッケージ構成のプランニング	92
論理ボリュームとファイルシステムのプランニング	93
NFS マウントされたファイルシステムのプランニング.....	94
拡張のプランニング.....	96
切り替え動作とフェイルオーバー動作の選択.....	96
汎用リソースを構成するパラメーター.....	97
汎用リソースの構成.....	97
シンプル/拡張汎用リソースのステータスと値の取得と設定.....	99

Serviceguard コマンドを使用したシンプル/拡張汎用リソースのステータスと値の取得	99
Serviceguard コマンドを使用したシンプル/拡張汎用リソースのステータスと値の設定	99
汎用リソースのオンライン再構成	100
パッケージの依存関係について	100
単純依存関係	100
単純依存関係の規則	101
単純依存関係のドラッグ規則	102
単純依存関係のガイドライン	104
拡張依存関係	105
排他的依存関係の規則	106
different_node と any_node の依存関係の規則	106
パッケージに障害が発生したときの動作	107
詳細情報	108
パッケージの重みについて	108
パッケージ重みとノードキャパシティ	108
重みとキャパシティの構成	108
単純な方法	108
例 1	109
留意すべき事項	109
包括的な方法	110
キャパシティの定義	110
重みの定義	111
規則とガイドライン	113
詳細情報	114
パッケージ重みとパッケージの優先順位および依存関係との相互作用	114
例 1	114
例 2	115
外部スクリプトについて	115
外部スクリプトでの Serviceguard コマンドの使用	117
パッケージがシャットダウンした原因の調査	117
last_halt_failed フラグ	118
クロスサブネットフェイルオーバーについて	118
アプリケーションの配備への影響	119
サブネットを越えてフェイルオーバーするパッケージの構成: 例	119
node_name の構成	119
monitored_subnet_access の構成	120
ip_subnet_node の構成	120
パッケージの構成: 次の手順	120
クラスター規模の変更の計画	120
5 HA クラスターの構成	122
システムの準備	122
Serviceguard のインストールとアップデート	122
Serviceguard のファイルの位置	122
Serviceguard のコマンドアクセスの有効化	123
ルートレベルアクセスの構成	123
未構成ノードへのルートアクセスの許可	123
他のノードのルートユーザーが認識されるようにする	124
identd について	124
名前解決の構成	124
名前解決サービスの障害からの保護	126
カーネル構成の一貫性	127
ネットワークタイムプロトコルの使用	127

チャンネルボンディングの構成 (Red Hat の場合).....	127
構成例.....	127
ネットワークの再起動.....	128
構成の表示.....	129
チャンネルボンディングの構成 (SUSE の場合).....	129
ネットワークの再起動.....	130
ロック LUN の設定.....	130
クォーラムサーバーの設定と実行.....	132
論理ボリュームインフラストラクチャの作成	132
ディスク情報の表示.....	133
パーティションの作成.....	133
ボリュームグループのアクティブ化保護を有効にする.....	135
ボリュームグループの構築:Smart Array クラスタストレージ (MSA 2000 シリーズ) の 例.....	136
ボリュームグループと論理ボリュームの構築.....	137
すべてのノードへの共有構成の配布.....	138
共有構成のテスト.....	138
ボリュームグループ構成データの保存	140
ブート時に vgscan が実行されないようにし、Serviceguard ボリュームグループがアク ティブにならないようにする.....	140
ディスクモニターのセットアップ.....	141
クラスタの構成.....	141
cmquerycl のオプション.....	141
処理の高速化.....	141
クラスタースト名のアドレスファミリの指定.....	142
ハートビートのアドレスファミリの指定	142
クラスタロックの指定.....	143
全ネットワークのプロローピング.....	143
ロック LUN の指定.....	143
クォーラムサーバーの指定.....	143
クロスサブネット情報の取得.....	144
ハートビートサブネットの識別.....	145
構成できるパッケージの最大数を指定する	145
MEMBER_TIMEOUT パラメーターの変更.....	146
クラスタへのアクセスの制御.....	146
用語についての注意事項.....	146
アクセスロールの動作.....	146
アクセスレベル.....	147
アクセス制御ポリシーの設定.....	148
ロールの競合.....	150
パッケージとクラスタのロール.....	151
クラスタ構成の確認	151
クラスタロック構成メッセージ.....	152
バイナリ構成ファイルの配布	152
稼働中のクラスタの管理.....	152
Serviceguard コマンドでのクラスタの動作チェック.....	152
自動起動機能のセットアップ	153
システムメッセージの変更	154
単一ノードクラスタの管理.....	154
単一ノードによる運用.....	155
identd の無効化.....	155
クラスタ構成の削除	155
6 パッケージとサービスの構成	157
パッケージモジュールの選択.....	157

パッケージのタイプ: フェイルオーバー、マルチノード、システムマルチノード	158
フェイルオーバーパッケージとマルチノードパッケージの違い	158
パッケージモジュールとパラメーター	159
ベースパッケージモジュール	159
オプションのパッケージモジュール	160
パッケージのパラメーターの説明	162
package_name	163
module_name	163
module_version	163
package_type	163
package_description	164
node_name	164
auto_run	164
node_fail_fast_enabled	164
run_script_timeout	165
halt_script_timeout	165
successor_halt_timeout	166
script_log_file	166
operation_sequence	166
log_level	166
failover_policy	166
failback_policy	167
priority	167
dependency_name	167
dependency_condition	168
dependency_location	168
weight_name、weight_value	168
monitored_subnet	169
monitored_subnet_access	169
monitored_subnet_access	169
ip_subnet	170
ip_subnet_node	171
ip_address	171
service_name	171
service_cmd	171
service_restart	172
service_fail_fast_enabled	172
service_halt_timeout	172
generic_resource_name	172
generic_resource_evaluation_type	173
generic_resource_up_criteria	173
vgchange_cmd	174
vg	174
ファイルシステムパラメーター	174
concurrent_fsck_operations	175
fs_mount_retry_count	175
fs_umount_retry_count	175
fs_name	176
fs_server	176
fs_directory	176
fs_type	176
fs_mount_opt	177
fs_umount_opt	177
fs_fsck_opt	177
pv	177

pev_	178
external_pre_script	178
external_script	178
user_host	178
user_name	178
user_role	179
従来のパッケージでのみ使われるその他のパラメーター	179
パッケージ構成ファイルの生成	179
準備作業	180
cmmakepkg の例	180
次の手順	181
構成ファイルの編集	181
パッケージ構成の検証と適用	184
クラスターへのパッケージの追加	184
ディスク監視構成の作成	184
7 クラスターとパッケージの管理	186
クラスターとパッケージのステータスの確認	186
cmviewcl コマンドによるクラスターとパッケージ状態の確認	186
パッケージの依存関係の表示	186
クラスターのステータス	186
ノードのステータスと状態	187
パッケージのステータスと状態	187
パッケージの切り替え属性	189
サービスのステータス	189
汎用リソースのステータス	189
ネットワークのステータス	190
フェイルオーバーポリシーとフェイルバックポリシー	190
クラスターとパッケージの状態の例	190
正常稼働中のステータス	190
クォーラムサーバーのステータス	191
パッケージ停止後のステータス	191
パッケージを別のノードへ移動した後のステータス	192
Package Switching (パッケージ切り替え) が Enabled に設定された後のステータス	193
ノード停止後のステータス	194
所有されていない (unowned) パッケージの情報の表示	194
クラスター構成要素のチェック	195
クラスターとノードの管理	196
全ノードが停止しているときのクラスターの起動	197
稼働中のクラスターへの、構成済みノードの追加	197
稼働中のクラスターからのメンバーノードの削除	198
Serviceguard コマンドによる、稼働中のクラスターからのメンバーノードの削除	198
クラスター全体の停止	198
クラスターの自動再起動	198
パッケージを実行した状態でのノードまたはクラスターの停止	199
提供されている機能	199
規則と制限事項	199
その他の注意点	200
ノードの停止とそのパッケージの切り離し	201
切り離されたパッケージの停止	202
クラスターの停止とそのパッケージの切り離し	202
例: ハートビートサブネットでの保守のためのクラスターの停止	202
パッケージとサービスの管理	203
パッケージの起動	203
依存関係のあるパッケージの起動	203

パッケージの停止	203
依存関係を持つパッケージの停止.....	204
フェイルオーバーパッケージの移動	204
パッケージ切り替え動作の変更	204
パッケージの保守: 保守モード.....	205
保守モードまたは部分起動保守モードでのパッケージの実行の特性	206
保守モードまたは部分起動保守モードでのパッケージの規則	206
部分起動保守モードの追加規則.....	207
保守モードまたは部分起動保守モードでのパッケージの依存関係規則	207
保守モードを使用した保守の実施.....	208
手順.....	208
部分起動保守モードを使用した保守の実施.....	208
手順.....	208
部分起動保守モードでのモジュールの除外.....	209
クラスタの再構成.....	209
クラスタの変更による影響のプレビュー.....	210
プレビュー可能な事象.....	210
Serviceguard Manager でのコマンドのプレビューモードの使用.....	211
cmeval の使用.....	212
停止したクラスタの再構成	212
稼働中のクラスタの再構成.....	213
稼働中のクラスタへのノードの追加	213
稼働中のクラスタからのノードの削除	214
稼働中のクラスタのクラスタネットワーク構成の変更.....	214
提供されている機能.....	214
留意事項.....	215
例: ハートビート LAN の追加.....	216
例: パッケージで使われるサブネットの削除.....	217
オンラインでのクラスタロック LUN 構成のアップデート.....	217
MAX_CONFIGURED_PACKAGES の変更	218
従来のパッケージの構成.....	218
従来のパッケージ構成の作成	218
Serviceguard Manager を使ったパッケージの構成	218
Serviceguard コマンドを使ったパッケージの構成	218
パッケージの構成手順.....	219
パッケージ構成ファイルの編集.....	219
パッケージ制御スクリプトの作成.....	221
パッケージ制御スクリプトのカスタマイズ	221
パッケージ制御スクリプトへのユーザー定義関数の追加	222
ユーザー定義関数への Serviceguard コマンドの追加	222
追加製品のサポート.....	223
パッケージ構成の確認.....	223
構成の配布.....	223
Serviceguard Manager による構成と制御スクリプトの配布.....	223
Linux コマンドによるパッケージ制御スクリプトのコピー.....	224
Linux コマンドによるバイナリ形式クラスタ構成ファイルの配布	224
クロスサブネットフェイルオーバーの構成.....	224
node_name の構成.....	224
monitored_subnet_access の構成.....	225
サブネット固有のパッケージ制御スクリプトの作成.....	225
nodeA と nodeB の制御スクリプトのエントリー.....	225
nodeC と nodeD の制御スクリプトのエントリー.....	225
パッケージの再構成.....	225
従来のパッケージからモジュラーパッケージへの移行.....	226
稼働中のクラスタでのパッケージ再構成	226

稼働中のパッケージで使用される外部スクリプトの名前変更または交換	226
停止しているクラスターでのパッケージの再構成	227
稼働中のクラスターへのパッケージの追加	227
稼働中のクラスターからのパッケージの削除	227
サービスの再起動カウンターのリセット	228
構成変更が可能なパッケージの状態	228
警告が発せられる変更	231
クラスターイベント発生時の対処	232
単一ノードによる運用	232
システム上の Serviceguard の削除	233
8 クラスターのトラブルシューティング	234
クラスターの運用テスト	234
パッケージマネージャーのテスト	234
クラスターマネージャーのテスト	235
ハードウェアの監視	236
ディスクの交換	236
ディスクアレイ内の故障したディスクドライブの交換	236
ロック LUN の交換	236
破局的な障害後の Persistent Reservation の取り消し	237
例	238
LAN カードの交換	238
障害が発生したクォーラムサーバーシステムの交換	239
トラブルシューティングの手掛かり	240
パッケージ IP アドレスの確認	240
システムログファイルの確認	241
システムログのエントリーの例	241
構成ファイルの確認	242
パッケージ制御スクリプトの確認	242
cmquerycl と cmcheckconf コマンドの使用	242
LAN 構成の確認	243
問題の解決	243
名前解決の問題	243
ネットワークとセキュリティの構成エラー	243
切り離されたパッケージの停止	243
一時的な状況によって引き起こされるクラスターの再編成	244
MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスターの再編成	244
システム管理エラー	245
パッケージ制御スクリプトのハングまたは障害	245
パッケージの移動エラー (従来のパッケージ)	247
ノード障害とネットワーク障害	247
クォーラムサーバーのトラブルシューティング	248
承認ファイルの問題	248
タイムアウトの問題	248
メッセージ	248
ロック LUN のメッセージ	248
Serviceguard Manager のトラブルシューティング	249
A 高可用性クラスターアプリケーションの設計	250
アプリケーション操作の自動化	250
ユーザーが故障停止の影響を受けないようにする	251
アプリケーションの起動/シャットダウン手順の定義	251
アプリケーションのフェイルオーバー速度の制御	251
非データファイルシステムの複写	252
ジャーナルファイルシステム (JFS) 使用の評価	252

データ損失の最小化	252
メモリベースのデータの使用とデータの量を最小限にする	252
ログのサイズを小さくする	252
ローカルデータの必要性の排除	253
再開可能なトランザクションの使用	253
チェックポイントの使用	253
チェックポイント回数と性能のバランス	253
複数サーバーの設計	254
複製データサイトの設計	254
複数のシステムで実行されるアプリケーションの設計	254
ノード固有情報の回避	255
十分な IP アドレスの獲得	255
同一システム上の複数インスタンスの生成	255
SPU ID または MAC アドレスの使用回避	255
アプリケーションへの固有の名前の割り当て	256
DNS の使用	256
uname(2) の慎重な使用	257
固定ポートへのバインド	257
再配置可能 IP アドレスへのバインド	257
connect() の前に bind() を呼び出す	257
各アプリケーションへの独自のボリュームグループの割り当て	258
SNA アプリケーションへの複数のあて先の使用	258
ファイルロックの回避	258
クライアント接続の復元	258
アプリケーション障害の処理	259
障害に強いアプリケーションの作成	260
アプリケーションの監視	260
計画的ダウンタイムの短縮	260
アプリケーションのアップグレードやパッチに必要な時間の短縮	260
段階的アップグレードの実行	261
リリース間でデータレイアウトを変更しない	261
オンラインでのアプリケーションの再構成の準備	261
保守作業の文書化	261
B HA アプリケーションと Serviceguard のインテグレーション.....	262
HA アプリケーションのインテグレーション用チェックリスト	262
単一システムでのアプリケーションの基本動作の定義	262
HA アプリケーションの複数のシステムへの組み込み	263
クラスターのテスト	263
C 未記入のプランニングワークシート	264
ハードウェア用ワークシート	264
電源用ワークシート	264
クォーラムサーバー用ワークシート	265
ボリュームグループと物理ボリューム用ワークシート	265
クラスター構成用ワークシート	266
パッケージ構成用ワークシート	266
パッケージ制御スクリプト用ワークシート (従来のパッケージ).....	267
D IPv6 ネットワークのサポート	268
IPv6 アドレスの種類.....	268
IPv6 アドレスのテキスト表現.....	268
IPv6 アドレスプレフィックス.....	269
ユニキャスト.....	269
IPv4 と IPv6 の互換性.....	269
IPv4 互換 IPv6 アドレス.....	269

IPv4 マップド IPv6 アドレス.....	269
集約可能グローバルユニキャストアドレス.....	270
リンクローカルアドレス.....	270
サイトローカルアドレス.....	270
マルチキャストアドレス.....	270
ネットワーク構成の制限事項.....	271
Linux での IPv6 の構成.....	272
Red Hat Linux での IPv6 の有効化.....	272
Red Hat Linux での恒久的 IPv6 アドレスの追加.....	272
Red Hat Linux での、恒久的 IPv6 アドレスによるチャンネルボンディングインターフェイスの構成.....	272
SUSE での恒久的 IPv6 アドレスの追加.....	273
SUSE での、恒久的 IPv6 アドレスによるチャンネルボンディングインターフェイスの構成.....	273
E Serviceguard Manager の使用.....	274
オンラインヘルプシステムについて.....	274
Serviceguard Manager の起動.....	274
シナリオ 1 - 単一クラスターの管理.....	274
シナリオ 2 - マルチクラスターの管理.....	276
F パラメーターの最大値と最小値.....	278
G 汎用リソース用の監視スクリプト.....	279
監視スクリプトの起動.....	279
監視スクリプトのテンプレート.....	281
H HP Serviceguard Contributed Toolkit.....	284
索引.....	285

出版履歴

表 1

出版日付	部品番号	版
2001 年 11 月	B9903-90008 (英語版: B9903-90005)	第 1 版
2002 年 11 月	英語版のみ (英語版: B9903-90012)	第 1 版
2002 年 12 月	B9903-90022 (英語版: B9903-90012)	第 2 版
2003 年 11 月	B9903-90034 (英語版: B9903-90033)	第 3 版
2005 年 2 月	B9903-90045 (英語版: B9903-90043)	第 4 版
2005 年 6 月	B9903-90047 (英語版: B9903-90046)	第 5 版
2006 年 8 月	B9903-90052 (英語版: B9903-90050)	第 6 版
2007 年 7 月	B9903-90057 (英語版: B9903-90054)	第 7 版
2008 年 3 月	B9903-90064 (英語版: B9903-90060)	第 8 版
2009 年 4 月	B9903-90070 (英語版: B9903-90068)	第 9 版
2009 年 7 月	英語版のみ (英語版: B9903-90073)	第 10 版
2012 年 6 月	701460-191 (英語版: 701460-001)	

最後の出版日と部品番号が最新版を表し、HP Serviceguard for Linux バージョン A.11.20.00 に適用されます。

出版日付は新しい版のリリース時に変更されます。内容の小さな変更に対しては、増刷の際に対応し、出版日の変更は行いません。マニュアルの部品番号は、大幅な技術的変更が行われた場合に改訂されます。

新版の作成は、記載内容の訂正またはドキュメント製品の変更に伴って行われます。

原典

本書は、『Managing HP Serviceguard A.11.20.00 for Linux』 (HP Part No. 701460-001) を翻訳したものです。

はじめに

本書は、HP ProLiant サーバー上の Linux オペレーティングシステムでの Serviceguard for Linux の構成方法と管理方法について説明します。本書は経験を積んだ Linux システム管理者を対象としています。(Serviceguard 固有でない Linux システムの管理作業については、システム管理ドキュメントと、使用している Linux ディストリビューションのマニュアルを参照してください。)

- ❶ **重要:** SUSE Linux Enterprise Server (SLES) は、Serviceguard A.11.20.00 バージョンではサポートされません。

内容は次のとおりです。

- 第1章 (18 ページ) では、Serviceguard クラスタとこのガイドを使用する方法を説明します。
- 第2章 (23 ページ) では、Serviceguard が使用するハードウェア構成の概要を説明します。
- 第3章 (29 ページ) では、Serviceguard のソフトウェア構成要素を説明し、各構成要素が Linux オペレーティングシステム内でどのように機能するかを示します。
- 第4章 (72 ページ) では、プランニングの過程を順を追って説明します。
- 第5章 (122 ページ) では、クラスタ構成の作成方法を説明します。
- 第6章 (157 ページ) では、高可用性パッケージの作成方法を説明します。
- 第7章 (186 ページ) では、基本的なクラスタ管理作業を説明します。
- 第8章 (234 ページ) では、クラスタのテスト方法とトラブルシューティングについて説明します。
- 付録 A (250 ページ) では、Serviceguard 環境で最適性能を実現するクラスタ対応アプリケーションを作成するためのガイドラインを示します。
- 付録 B (262 ページ) では、既存のアプリケーションを Serviceguard for Linux に統合する手順を説明します。
- 付録 C (264 ページ) には、Serviceguard の構成を検討する際に使用するワークシート一式があります。
- 付録 D (268 ページ) には、IPv6 に関する情報があります。
- 付録 E (274 ページ) には、Serviceguard Manager の紹介があります。
- 「パラメーターの最大値と最小値」 (278 ページ) では、サポートされる Serviceguard パラメーターの範囲について説明しています。
- 付録 G 「汎用リソース用の監視スクリプト」では、汎用リソース用の監視スクリプトのテンプレートを提供します。
- 「HP Serviceguard Contributed Toolkit」 (284 ページ) では、一般的なアプリケーションを Serviceguard に容易に組み込めるようにする一連のツールについて説明します。

関連マニュアル

詳細は、以下の最新版のマニュアルを参照してください。

- 『HP Serviceguard for Linux リリースノート』
- 『HP Serviceguard Quorum Server Release Notes』
- 『クラスタによるハイアベイラビリティの構築 - オープンシステムでミッションクリティカル環境を実現する -』 ピアソン・エデュケーション (ISBN: 4-89471-545-7) <http://h50146.www5.hp.com/products/software/oe/hpux/developer/book.html?>

[jumpid=reg_R1002_JPJA](#) (日本語版)、『Clusters for High Availability: a Primer of HP Solutions』 Second Edition.HP Press, 2001 (ISBN 0-13-089355-2)

当社のハイアベイラビリティ (HA) システムのドキュメントに関する Web ページへは、次の URL でアクセスしてください。

<http://www.hp.com/go/linux-serviceguard-docs>

『HP Serviceguard for Linux Configuration Guide』には、サポートされる構成についての最新情報が記載されています。サポートされるハードウェアおよび Linux ディストリビューションの最新情報は、『HP Serviceguard for Linux Certification Matrix』でご確認ください。この 2 つのドキュメントは次の URL で入手できます。

<http://www.hp.com/info/sglx> -> "Configuration guide"および"Certification Matrix"

問題の報告

ソフトウェアやマニュアルに問題がある場合は、当社の営業担当またはサポート担当者にお問い合わせください。

1 Serviceguard for Linux の概要

本章では、Serviceguard for Linux の概要を説明します。また、本書の各章に記載されている情報についても説明します。この章の内容は以下のとおりです。

- Serviceguard for Linux について
- Serviceguard Manager の使用 (20 ページ)
- 構成の手順 (21 ページ)

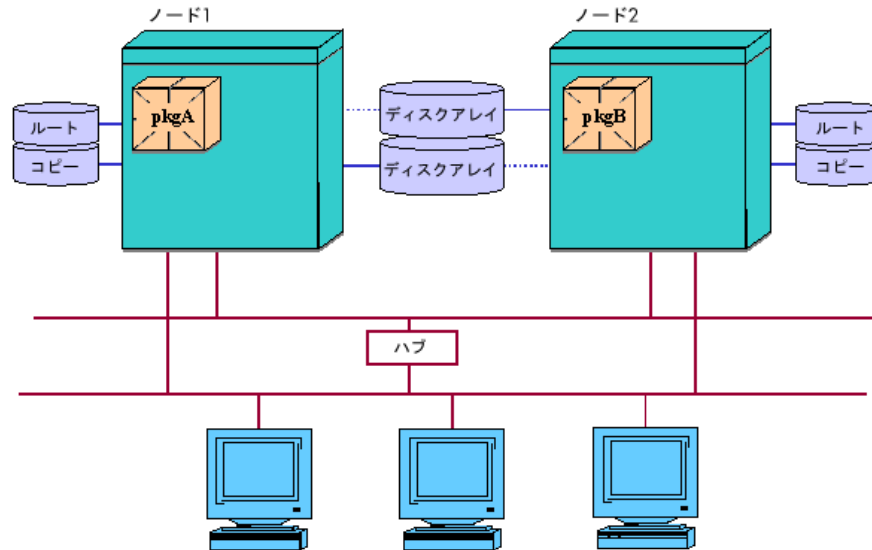
Serviceguard クラスターの設定準備ができている場合は、[第4章 \(72 ページ\)](#) に進んでください。詳しい設定手順については、[第5章 \(122 ページ\)](#) を参照してください。

Serviceguard for Linux について

Serviceguard for Linux によって、HP ProLiant サーバーの高可用性クラスターの作成が可能になります。高可用性 コンピューターシステムにより、ハードウェアやソフトウェアの障害が発生した場合も、アプリケーションサービスを続行できます。高可用性システムは、ソフトウェア障害だけでなく、システム演算処理装置 (SPU)、ディスク、またはローカルエリアネットワーク (LAN) といった構成要素の障害からもユーザーを守ります。ある構成要素に障害が発生した場合、代理の構成要素がその処理を引き継ぎます。Serviceguard と他の高可用性サブシステムは、このような構成要素間の移動を調整します。

Serviceguard クラスターとは、HP ProLiant サーバー (ノードと呼ばれるホストシステム) のネットワーク接続された集合を意味します。クラスターは、ソフトウェアおよびハードウェアを十分に冗長化することによって、単一障害点 によるサービスの大きな中断を防ぎます。アプリケーションサービス (個々の Linux プロセス) は、パッケージにグループ化されます。サービス、ノード、ネットワークなどのリソースの 1 箇所で障害が発生した場合、Serviceguard はパッケージの制御をクラスター内の別のノードに自動的に移動することにより、サービス中断を最小限に抑え使用可能な状態を維持します。

図 1 代表的なクラスター構成



図では、ノード 1(2 つの SPU の一方) はパッケージ A を実行していて、ノード 2 はパッケージ B を実行しています。各パッケージは、それぞれ関連するディスクのグループを持ち、これらのディスクグループは、パッケージのアプリケーションが必要とするデータとそのデータのコピーを持っています。この 2 つのノードはともに、ディスクアレイに物理的に接続されていますが、ディスクグループのデータには一度に 1 つのノードしかアクセスできません。図では、ノード 1 は上側の 2 つのディスクへの排他的アクセス権を持ち (実線)、ノード 2 は上側のディスクへ接続されているがアクセス権は持っていないことが示されています (点線)。同様に、ノード 2 は下側の 2 つのディスクへの排他的アクセス権を持ち (実線)、ノード 1 は下側のディスクへ接続されているがアクセス権は持っていないことが示されています (点線)。

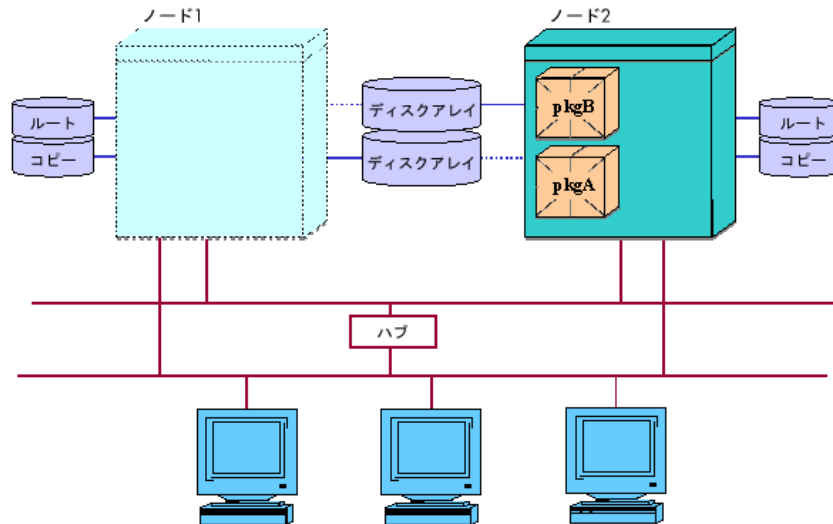
ディスク障害が発生した場合でも、ディスクアレイにより冗長性が確保されます。さらに、ノード 1 とノード 2 に接続されているディスクには、合計 4 つのデータバスを使用できます。この構成では、各パッケージが個別のバスを使用しているので、最大の冗長性と入出力性能が得られます。

また、各ノードで冗長 LAN インターフェイスが利用できるようにネットワークハードウェアが接続されていることに着目してください。Serviceguard は TCP/IP ネットワークサービスを使って、クラスター内のノード間で信頼性の高いデータ通信を実現します。データ通信には、ハードビートメッセージ (クラスター運用の根幹をなす、動作中のノードからの信号) の送信も含まれます。上記以外の種類のノード間通信には、TCP/IP サービスも使用されます。(ハードビートの詳細は、「[Serviceguard のソフトウェア構成要素](#)」(29 ページ) を参照してください。)

フェイルオーバー

完全に稼働した状態の Serviceguard クラスターは通常、各ノード上でパッケージが実行されている間、クラスターの構成要素の状態を単に監視しています。Serviceguard クラスター内で動作しているホストシステムを、アクティブノードと呼びます。パッケージを作成するときに、1 つのプライマリノードと、1 つ以上の引き継ぎノードを指定します。ノードまたはネットワーク通信に障害が発生した場合、Serviceguard はパッケージの制御を、次に使用可能な引き継ぎノードに移すことができます。この状況を、[図 2 \(20 ページ\)](#) に示します。

図 2 フェイルオーバー後の代表的なクラスター



制御を移行した後、引き継ぎノードが動作している間は通常、パッケージは引き継ぎノード上にあります。ただし必要に応じて、プライマリノードがオンラインに復帰した時点で直ちにパッケージをプライマリノードに戻すよう構成できます。また、時期を見計らって手動でプライマリノードに制御を戻すこともできます。

図 2 (20 ページ) では示していませんが、クラスターへの電源の接続も重要です。クラスターからすべての単一障害点を取り除くには、ノード、ディスク、ミラー化ディスクの障害回避に必要な数だけ、個別の電源システムを用意することが必要です。また、各電源システムは無停電電源装置 (UPS) で保護することが必要です。詳細は、「[電源のプランニング](#)」(76 ページ) の項を参照してください。

Serviceguard は、他のハイアベイラビリティ製品 (ディスクアレイなど) と連動するように設計されています。ディスクアレイでは、さまざまな RAID レベルでデータを保護できます。また、HP がサポートする UPS は、電源故障に関連する障害を取り除きます。最高レベルの可用性を実現するために、これらの製品を Serviceguard と併用することをお勧めします。

Serviceguard Manager の使用

注記: 詳細は、[付録 E \(274 ページ\)](#) と、最新版の Serviceguard リリースノートで「Serviceguard Manager について」の項を参照してください。Serviceguard Manager の機能の詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] にある『Serviceguard/Serviceguard Manager Plug-in Compatibility and Feature Matrix』と最新版のリリースノートを参照してください。

Serviceguard Manager は、Serviceguard のグラフィカルユーザーインターフェイスです。Web ベースの HP System Management Homepage (HP SMH) の「プラグイン」として利用できます。Serviceguard Manager から、Serviceguard クラスターの監視、管理、構成を行うことができます。

- クラスター、ノード、およびパッケージのプロパティ、ステータス、および警告を参照することができます。
- クラスター、クラスターノード、およびパッケージの起動や停止といった管理作業を行うことができます。

- クラスターやそのパッケージを作成したり変更することができます。

Serviceguard Manager の使用についての概要は、[付録 E \(274 ページ\)](#) を参照してください。Serviceguard Manager の機能の詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] にある『Serviceguard/Serviceguard Manager Plug-in Compatibility and Feature Matrix』と最新版のリリースノートを参照してください。

Serviceguard Manager を使ったクラスターの監視

Serviceguard Manager のメインページから、クラスター、ノード、およびパッケージのステータスおよび警告を参照できます。また、クラスター、ノード、およびパッケージの構成および警告の詳細を表示することもできます。

Serviceguard Manager を使ったクラスターの管理

アクセス制御ポリシーで許可されていれば、クラスター、ノード、パッケージを管理することができます。

- クラスター: 停止、起動
- クラスターノード: 停止、起動
- パッケージ: 停止、起動、ノード間の移動、ノード切り替えフラグやパッケージ切り替えフラグのリセット

Serviceguard Manager を使ったクラスターの構成

Serviceguard Manager では、クラスターとパッケージを構成することができます。クラスター管理権限を持っている必要があります。

Serviceguard Manager の起動

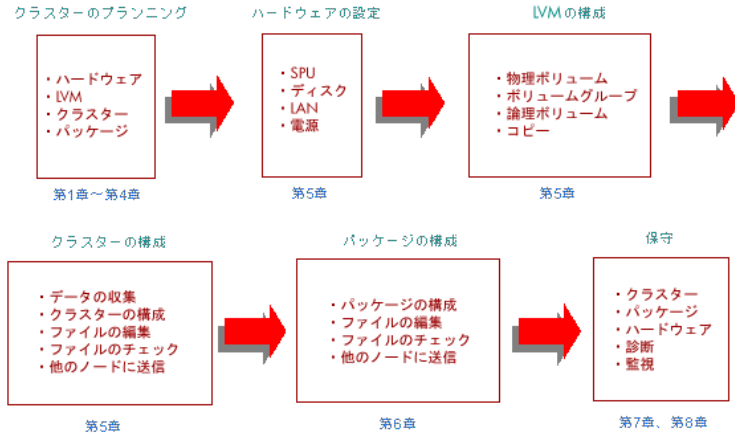
Serviceguard Manager を起動するには、使用しているバージョンの Serviceguard for Linux のリリースノートに記載されている手順に従います。次に、クラスター、ノード、パッケージのいずれかを選択し、[Serviceguard Manager] バナーの下にあるドロップダウンメニューを使って、必要なタスクを開きます。

Serviceguard Manager の組み込みヘルプでは手順が説明されています。本書では、Serviceguard Manager でどのような作業ができるのかを説明します。

構成の手順

本書では、Serviceguard を使って HA (ハイアベイラビリティ) クラスターを作成する方法について説明します。各作業を [図 3](#) に示します。

図 3 Serviceguard クラスターの構成作業



構成を開始する前に、構成に必要なすべてのデータを収集しておくことをお勧めします。データ収集のヒントについては、[第4章 \(72 ページ\)](#) を参照してください。

2 Serviceguard for Linux のハードウェア構成について

本章では、サーバーのハードウェア構成要素が Serviceguard for Linux とどのように連携して動作するかについて、概要を説明します。説明する項目は以下のとおりです。

- 冗長クラスター構成要素
- 冗長ネットワーク構成要素 (23 ページ)
- 冗長ディスクストレージ (27 ページ)
- 冗長電源 (28 ページ)

Serviceguard ソフトウェア構成要素の詳細については、第 3 章「Serviceguard のソフトウェア構成要素」を参照してください。

冗長クラスター構成要素

代表的なクラスターは、高いレベルの可用性を提供するために、冗長なシステム構成要素を使用します (2 つ以上の SPU と 2 つ以上の独立したディスクなど)。冗長性を持たせることにより、単一障害点を取り除くことができます。一般的に、冗長性が増すと、障害時でもアプリケーション、データ、およびサポートされているサービスが使用可能である可能性が高まります。ただしこのためには、ハードウェアの冗長性だけでなく、障害発生時に、別の SPU またはネットワークへアプリケーションを移行させるためのソフトウェアが必要になります。Serviceguard は、以下の機能を提供します。

- LAN 障害に対しては、Linux のボンディング機能による待機 LAN を使用するか、Serviceguard がパッケージを他のノードへ移動します。
- SPU で障害が発生した場合、アプリケーションは障害の発生した SPU から機能している SPU に、最短の時間で自動的に移動されます。
- ソフトウェアで障害が発生した場合、アプリケーションは最短の中断で同じノード上または別のノード上で再起動されます。

また、Serviceguard には、元の SPU を停止させてシステム管理、保守、バージョンのアップグレードを行うために、アプリケーションの制御を別の SPU へ容易に移行できるという利点もあります。

Serviceguard for Linux クラスターでサポートされるノード数は最大 4 台ですが、実際数はストレージ構成によって異なります。たとえば、パッケージが FibreChannel 接続を介してデータにアクセスする場合は、16 台のノードのいずれかへフェイルオーバーするように構成できますが、SCSI ディスクアレイのデータにアクセスする場合は、通常、4 台のノードに限定されます。

共有ストレージのデータを使用しないパッケージについては、ディスクの種類を問わずクラスター内に構成されているすべてのノード (最大 16 台のノード) へフェイルオーバーするように構成できます。たとえば、パッケージが単にローカルノードの実行ファイルを実行するだけで、ローカルデータしか使用しない場合は、クラスター内のすべてのノードへフェイルオーバーするように構成できます。

冗長ネットワーク構成要素

ネットワークで発生する単一障害点を取り除くには、クラスターノードがアクセスする各サブネットが冗長ネットワークインターフェイスを持つ必要があります。また、ケーブル障害を防止するために、冗長ケーブルも必要です。各インターフェイスカードは、個別のケーブル、ハブまたはスイッチに接続されます。

チャネルボンディングという機能により、複数のネットワークインターフェイスで IP アドレスを共有させることができます。「[チャネルボンディングの構成 \(Red Hat の場合\)](#)」(127 ページ) または「[チャネルボンディングの構成 \(SUSE の場合\)](#)」(129 ページ) を参照してください。

Serviceguard は、ノードごとに最大 30 個のネットワークインターフェイスをサポートしています。ここでいうインターフェイスは、`ifconfig` の出力でプライマリインターフェイスとして表示されるものと定義されるため、合計 30 個は、物理 LAN インターフェイスとボンディングインターフェイスを組み合わせたものとなります。(1 つのノードには、このようなインターフェイスを 30 個以上持たせることができますが、クラスター構成に組み込むことができるのは、30 個だけです。)

規則と制限事項

- 1 つのサブネットを、同じノード上の複数のネットワークインターフェイス (NIC) に構成することはできません。
- クラスターノード間の通信に使用できるサブネットの場合、同じネットワークインターフェイスを使用して同じノードに構成された複数のサブネットを経路指定することはできません。
- IPv4 サブネットの場合、Serviceguard は同じ LAN インターフェイス上での複数のサブネットはサポートしていません。
 - IPv6 の場合、Serviceguard は LAN インターフェイスごとに最大 2 つのサブネット (サイトローカルとグローバル) をサポートしています。
- Serviceguard は、同じブリッジネットワーク上での複数のサブネットをサポートしています (ノードレベルとクラスターレベルの両方に適用されます)。
- Serviceguard は、IP アドレス自体が **定常 IP アドレス** としてクラスターにすぐに構成されない限り、Serviceguard クラスターに構成されているネットワークインターフェイスへの IP アドレスの追加に、`ifconfig` などのネットワークツールを使用することはサポートしていません。

△ **注意:** Serviceguard ネットワークインターフェイスに定常 IP アドレス以外のアドレスを構成すると、Serviceguard が割り当てた再配置可能パッケージ IP アドレスと競合する可能性があります。「[定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット](#)」(54 ページ) を参照してください。

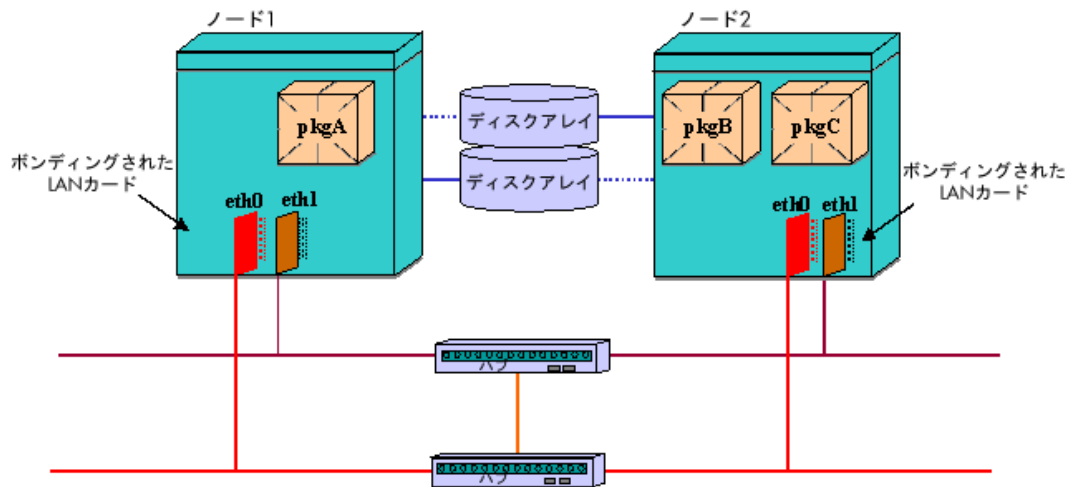
- 同様に、Serviceguard は、クラスターに構成されている IP アドレスの移動または再構成に、ネットワークツールを使用することはサポートしていません。
Serviceguard による構成の表示は実際とは異なっているため、この操作を行うと予測できない結果につながります。

注記: クロスサブネット構成を使用する場合は、このような構成にのみ適用される[制限事項](#) (26 ページ) も参照してください。

冗長イーサネット構成

イーサネット構成での冗長なネットワーク構成要素の使用例を [図 4](#) に示します。

図 4 冗長 LAN



Linux 構成では、イーサネットセグメントへの接続に冗長ハブまたはスイッチを使う、対称 LAN 構成を強くお勧めします。ソフトウェアボンディング構成では、アクティブインターフェイスが同じハブやスイッチに接続されている状態で、各ノードの構成を同じにする必要があります。

クロスサブネット構成

Serviceguard A.11.18 以降では、ルーターで結合された複数のサブネットを、クラスターハートビートとデータの両方で利用するように構成することができます。この構成では、一部のノードがあるサブネットを利用し、別のノードが別のサブネットを利用します。

クロスサブネット構成では、以下のことが可能です。

- あるサブネット上のノードから別のサブネット上のノードへのパッケージの自動的なフェイルオーバー
- 複数のサブネットにまたがるクラスターハートビート

構成作業

クラスターとパッケージの構成作業に以下の影響があります。

- 構成済みまたは構成可能なノードとサブネットをルーターにまたがって検出するためには、`cmquerycl` に `-w full` オプションを指定する必要があります。
- サブネットにまたがったパッケージのフェイルオーバーを可能にするために、パッケージ構成ファイルで以下の 2 つの新しいパラメーターを設定する必要があります。
 - `ip_subnet_node` - サブネットが構成されているノードを示します。

- `monitored_subnet_access` - サブネットがすべてのノード上で構成されているか (FULL)、一部のノードでのみ構成されているか (PARTIAL) を示します。

(従来のパッケージについては、「[クロスサブネットフェイルオーバーの構成](#)」(224 ページ) を参照してください。)

- パッケージ構成ファイルで `node_name` にワイルドカード (*) を使用することはできません。ワイルドカードを使用すると、同じサブネットのノードへのフェイルオーバーが可能であっても、パッケージがサブネットを越えてフェイルオーバーする可能性があります。サブネットを越えるフェイルオーバーは、同じサブネットでのフェイルオーバーよりも時間がかかる場合があります。ノードは、ワイルドカードを使用せずに、優先順位の高い順にリストしてください。
- 各サブネットに IP 監視を構成する必要があります。「[LAN インターフェイスの監視と障害の検出: IP レベル](#)」(59 ページ) を参照してください。

制限事項

以下の制限があります。

- クラスターのすべてのノードが、同じネットワークドメインに属している必要があります (すなわち、完全修飾ドメイン名のドメイン部分が同じでなければなりません)。
- 各ノードは、IP レベルで完全に接続されている必要があります。
- 各クラスターノードで最低 2 つのハートビートパスが構成されている必要があります。
- ハートビートネットワークの遅延時間が 200 ミリ秒未満である必要があります。
- 各ノードの各ハートビートサブネットでは、別のノード上のハートビートサブネットへ、物理的に異なる経路を指定する必要があります。すなわち、ハートビートパスが物理的に独立している必要があります。
 - ハートビートは静的に経路指定する必要があります。すなわち、異なるパスを通じてハートビートを経路指定するためには、各ノード上で静的な経路エントリが構成されている必要があります。
 - 1 台のルーターで障害が発生しても、両方のハートビートに同時に影響が出ないようにする必要があります。
- IPv6 ハートビートサブネットは、クロスサブネット構成ではサポートされません。
- IPv6 専用モードおよび混合モードは、クロスサブネット構成ではサポートされません。これらのモードの詳細は、「[ホスト名アドレスファミリーについて: IPv4 専用、IPv6 専用、および混合モード](#)」(79 ページ) を参照してください。
- この環境でアプリケーションを配備する際には、慎重に検討する必要があります。「[アプリケーションの配備への影響](#)」(119 ページ) を参照してください。
- 対象となるノードで「hostname LAN」のステータスが DOWN になっている場合、`cmrunnode` は失敗します (「ホスト名 LAN」とは、そのノードのホスト名が解決される IP アドレスが構成されているパブリック LAN のことを指します)。
- `monitored_subnet` に対し、パッケージ構成ファイルで、`monitored_subnet_access` に PARTIAL が設定されている場合、そのパッケージの `node_name` リスト中の 1 台以上のノードでそのサブネットが構成されている必要があります。逆に、このパッケージに対して監視されているすべてのサブネットで PARTIAL アクセスが構成されている場合、`node_name` リストの各ノードでは、これらのサブネットのうち 1 つ以上が構成されている必要があります。
 - 他の構成と同様に、ノード上に構成され、パッケージ構成ファイル中で監視対象サブネットとして指定されているサブネットが動作状態でない場合、パッケージはノード上で開始されません。

注記: すべてのクラスターネットワーク構成に適用される規則と制限事項 (24 ページ) も参照してください。

詳細情報

クロスサブネット環境での、クラスターとパッケージ構成の詳細については、「クロスサブネットフェイルオーバーについて」(118 ページ)、「クロスサブネット情報の取得」(144 ページ)を、従来のパッケージに関しては「クロスサブネットフェイルオーバーの構成」(224 ページ)を参照してください。

以下のアドレスにあるホワイトペーパー **Technical Considerations for Creating a Serviceguard Cluster that Spans Multiple IP Subnets** も参照してください。このホワイトペーパーでは、サポートされる構成について例を挙げて説明しています。また、間違えやすい構成についても説明しています。

- ① **重要:** クロスサブネットトポロジは単一サイトで実装することができますが、これは主に長距離クラスターで使用されます。このようなクラスターの詳細は、<http://www.hp.com/go/linux-serviceguard-docs> にある『HP Serviceguard Extended Distance Cluster for Linux Deployment Guide』を参照してください。

冗長ディスクストレージ

クラスターのノードはそれぞれルートディスクを持っています。ただし、構成されているパッケージに関連するデータやプログラムに複数のノードがアクセスできるように、各ノードは異なる他のディスクにも物理的に接続されていることもあります。このようなアクセスは、Logical Volume Manager (LVM) により提供されます。一度に複数のノードがボリュームグループを起動することはできません。ただし、パッケージを移動した場合は、このボリュームグループは、引き継ぎノードによって起動できます。

注記: リリース A.11.16.07 より、Serviceguard for Linux で HP-UX の排他アクティブ化と同等の機能が提供されるようになりました。この機能は、LVM2 のホストタグに基づいており、LVM2 を正式にサポートしている Linux ディストリビューションでのみ利用できます。

パッケージが所有するボリュームグループ内のディスクはすべて、そのパッケージの元のノードとすべての引き継ぎノードに接続されなければなりません。

Serviceguard for Linux クラスターの共有ディスクストレージは、ディスクアレイが提供しています。これらのシステムには、冗長電源があり、また複数のノードへ接続することができます。ディスクアレイは RAID モードを使って冗長性を提供します。

サポートされているディスクインターフェイス

2 つ以上のノード (共有データディスク) に接続されるディスクでは、以下のインターフェイスを使用できます。

- FibreChannel

マルチパス機構の構成については、「ストレージのマルチパス」(75 ページ)を参照してください。

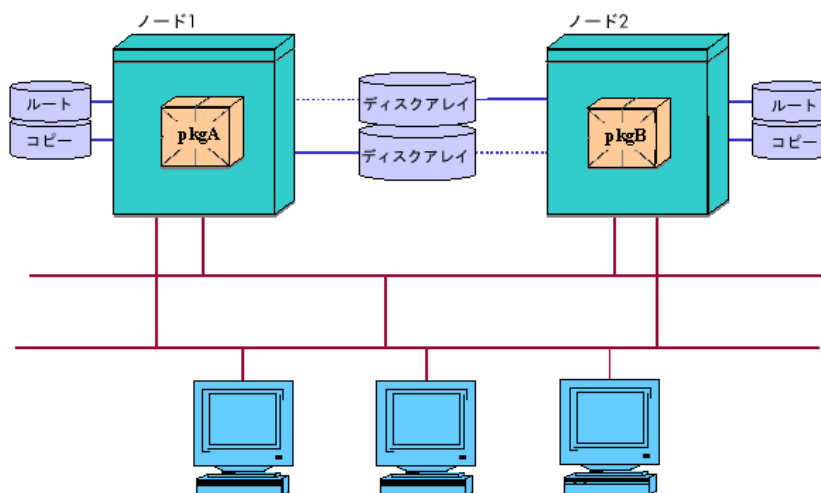
ディスクモニター

ディスクの監視を構成し、パッケージがモニターに依存するように構成することができます。パッケージごとに、そのパッケージによってアクティブ化されるディスクを監視するパッケージサービスを定義します。1 つのノード上でディスク障害が発生すると、モニターはパッケージを停止し、同じディスクを使用できる別のノードにフェイルオーバーします。

ディスク構成例

2 つノードを持つクラスターの例を図 5 に示します。各ノードには、ミラー化された 1 つのルートディスクと、プライマリノードとして設定された 1 つのパッケージがあります。各ノードが他のノードからパッケージを取り入れられるように、ノードごとにリソースが割り当てられています。各パッケージには 1 つのディスクボリュームグループが割り当てられ、ボリュームグループ内の論理ボリュームはミラー化されています。

図 5 高可用性を実現するミラー化ディスクの接続



冗長電源

ノードとディスクにバッテリーバックアップ機能を付けると、ハードウェアの可用性が拡張できます。当社がサポートする UPS (無停電電源装置) は、瞬間的な電源損失を保護できます。

ディスクアレイのコピーが異なる電源系統に接続されるような方法で、ディスクを電源に接続する必要があります。ブートディスクは、対応するノードと同じ電源系統から電源をとる必要があります。フォールムサーバーシステムの電源には、クラスターのノードとは別の電源を使用しなければなりません。クラスター用の電源、ディスク、および LAN ハードウェアのレイアウトについての詳細は、当社の担当者にお問い合わせください。

3 Serviceguard のソフトウェア構成要素

本章では、Serviceguard のソフトウェア構成要素の動作について概要を説明します。この章の内容は以下のとおりです。

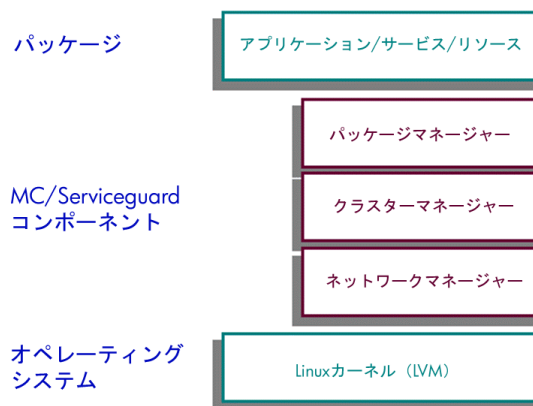
- Serviceguard のアーキテクチャー
- クラスタマネージャーの動作 (33 ページ)
- パッケージマネージャーの動作 (38 ページ)
- パッケージの動作 (47 ページ)
- ネットワークマネージャーの動作 (54 ページ)
- データストレージ用のボリュームマネージャー (64 ページ)
- 障害への応答 (68 ページ)

Serviceguard クラスターの設定準備ができている場合は、本章を省略して第 4 章「HA クラスターのプランニングと文書化」に進んでください。

Serviceguard のアーキテクチャー

下図に Serviceguard for Linux が使用する主なソフトウェア構成要素を示します。本章ではこれらの構成要素について詳しく説明します。

図 6 Linux 上の Serviceguard のソフトウェア構成要素



Serviceguard のデーモン

Serviceguard for Linux では、以下のデーモンが使われます。

- cmclconfd - 構成デーモン
- cmcld - クラスタデーモン
- cmnetd - Network Manager デーモン

- cmlogd - クラスターシステムログデーモン
- cmdisklockd - クラスターロック LUN デーモン
- cmresourced - Serviceguard 汎用リソースアシスタントデーモン
- cmserviced - サービスアシスタントデーモン
- qs - クォーラムサーバーデーモン
- cmlockd - ユーティリティデーモン
- cmsnmpd - クラスター SNMP サブエージェント (オプションで実行)
- cmwbemd - WBEM デーモン
- cmproxyd - プロキシデーモン

これらのデーモンのログは、Linux のシステムログファイルに記録されます。クォーラムサーバーデーモンのログはユーザーが指定したログファイルに記録されます。たとえば、Red Hat では /usr/local/qs/log/qs.log ファイル、SUSE では /var/log/qs/sq.log ファイルなどです。

注記: ファイル cmcluster.conf には、以下の各項のパス名で使用されている \$SGCONF、\$SGROOT、\$SGLBIN などのシンボル参照を解決するマッピングが含まれています。詳細は、「[Serviceguard のファイルの位置](#)」(122 ページ)を参照してください。

構成デーモン: cmclconfd

このデーモンは、クラスター内のすべてのノードから情報を集めるために、Serviceguard コマンドで使用されます。ネットワークおよびボリュームグループ情報などの構成情報を集めます。また、クラスターのバイナリ構成ファイルをクラスター内のすべてのノードに配布します。このデーモンはインターネットデーモン xinetd(1M) によって起動されます。

パラメーターは、/etc/xinetd.d/hacl-cfg ファイルと /etc/xinetd.d/hacl-cfgudp ファイルにあります。このデーモンへのパスは \$SGLBIN/cmclconfd です。

クラスターデーモン: cmcld

このデーモンは、Serviceguard クラスター内の他のノード上の cmcld デーモンにハートビートメッセージを送って、クラスターメンバーシップを決定します。このデーモンはリアルタイム優先順位で動作し、メモリ内にロックされます。cmcld デーモンは、カーネルのハングを検出するために使用されるカーネルの セーフティタイマーを設定します。このタイマーが cmcld によって定期的のリセットされない場合は、カーネルがシステムをリブートします。システムリブートは、cmcld がクラスターのメンバーの過半数と通信できない場合に発生します。また、cmcld が予期せず終了した場合、中断した場合、一定の時間実行できず、カーネルタイマーを更新できない場合(カーネルのハングを意味する)にも発生します。セーフティタイマーの時間切れによってシステムリセットが行われる前に、syslog およびカーネルのメッセージバッファにメッセージが書き込まれ、可能な場合には、システムダンプが実行されます。

セーフティタイマーの持続時間は、クラスター構成パラメーター `MEMBER_TIMEOUT`、およびクォーラムサーバーやクラスターロックを使用しているかどうか(とロックの種類)、スタンバイ LAN が構成されているかどうかなどのクラスター構成の特性に依存します。

詳細は、「[ノードがタイムアウトしたときの動作](#)」(68 ページ)を参照してください。

`MEMBER_TIMEOUT` の設定については、[クラスター構成のパラメーター](#) (82 ページ)を参照してください。トラブルシューティングについては、「[MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスターの再編成](#)」(244 ページ)を参照してください。

cmcld は、Serviceguard パッケージも管理し、それらの実行場所と起動時刻を決定します。このデーモンへのパスは \$SGLBIN/cmcld です。

注記: Serviceguard の 2 つの中心的構成要素 (パッケージマネージャー、クラスターマネージャー) は cmcld デーモンの一部として動作します。このデーモンは、優先順位 94 として、SCHED_RR クラスで動作します。他のプロセスを、このデーモンより高いリアルタイム優先順位とすることはできません。

ログデーモン: cmlogd

cmlogd は、システムログファイルにメッセージを書き込むために cmcld で使用されます。cmcld でシステムログファイルに書き込まれるメッセージは、cmlogd により書き込まれます。これは、syslog に書き込む際の遅延が cmcld のタイミングに影響を及ぼさないようにするためです。このデーモンへのパスは \$SGLBIN/cmlogd です。

Network Manager デーモン: cmnetd

このデーモンはクラスターネットワークの稼働状態を監視します。またこのデーモンは、IPv4 アドレスと IPv6 アドレスの両方に対して、再配置可能パッケージ IP の追加と削除を行います。

ロック LUN デーモン: cmdisklockd

ロック LUN が使用される場合、cmdisklockd はクラスター内の各ノード上で実行され、クラスターの再編成中に必要となった場合にタイブレークサービスを提供します。このデーモンは、ノードがクラスターに追加される際に cmcld によって起動されます。このデーモンへのパスは \$SGLBIN/cmdisklockd です。

サービスアシスタントデーモン: cmserviced

このデーモンは、クラスターデーモン cmcld で必要なスクリプトやプロセスを起動 (fork して exec) します。このデーモンでは、2 種類の fork を実行します。

- パッケージの実行スクリプトおよび停止スクリプトの実行
- サービスの起動

サービスの場合、cmcld は、サービスプロセスを監視し、サービスの再試行回数に応じて、cmsrvassistd を使用してサービスを再起動するか、パッケージを停止することによりそのパッケージを使用可能な代替ノードに移動します。このデーモンへのパスは \$SGLBIN/cmserviced です。

汎用リソースアシスタントデーモン: cmresourced

このデーモンは、パッケージの一部として構成される汎用リソースのステータスと値の設定および取得を行い、リソースの可用性に基づいてパッケージの可用性を決定します。

汎用リソースは、カスタム定義のモニターを Serviceguard に統合することを可能にします。このデーモンは、リソースのステータスの取得および設定について、制御を容易にし柔軟性を高めます。また、選択肢も増やします。

汎用リソースデーモンは、Serviceguard コマンドの cmgetresource(1m) および cmsetresource(1m) がパッケージで構成されるシンプル/拡張汎用リソースのステータスと値を取得または設定するために使用する、ノードのローカルデーモンです。このデーモンは、cmcld が動作しているすべてのノードで動作します。

クォーラムサーバーデーモン: qs

クォーラムサーバーの使用は、クラスター再編成時に、タイブレークを行い、クォーラムを確立するための方法の 1 つです。これ以外の方法としては、クラスターロックを使う方法があります。その他、ロック LUN を使用する方法もあります。[「スプリットブレインの問題を回避するクラスター定足数」 \(34 ページ\)](#) と、それに続く各項を参照してください。

クォーラムサーバーを使用する場合は、クラスター外部のシステムで実行します。通常は、/etc/inittab から respawn オプション付きで起動されるため、異常終了したり抹消さ

れた場合は、自動的に再起動されます。また、サービス対象ではないクラスターで Serviceguard パッケージとして構成することもできます。詳細は、[図 9 \(37 ページ\)](#) を参照してください。

クラスターのメンバーはすべて、クォーラムサーバーへの接続を確立し、保持します。クォーラムサーバーが終了すると、Serviceguard ノードはこの事実を検出し、定期的にサーバーに再接続しようとしています。クォーラムサーバーがダウンしている間にクラスターの再編成が発生し、タイブレークが必要となる場合、再編成は失敗し、すべてのノードが停止します (システムリセット)。このためできるだけ早急にクォーラムサーバーを元の状態に戻すことが重要です。

Quorum Server ソフトウェアとその機能の詳細、および Quorum Server を Serviceguard パッケージとして構成する手順については、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にある『HP Serviceguard Quorum Server Release Notes』の最新版を参照してください。「[クラスターロックとしてのクォーラムサーバーの使用](#)」(36 ページ) も参照してください。

このデーモンへのパスは以下のとおりです。

- SUSE の場合: /opt/qs/bin/qs
- Red Hat の場合: /usr/local/qs/bin/qs

ユーティリティデーモン: cmlockd

cmcltd が実行されている各ノード上で実行されます。このデーモンは、アクティブなクラスターリソースロックと、保留中のクラスターリソースロックを維持します。

クラスター SNMP エージェントデーモン: cmsnmpd

このデーモンは、SNMP Master Agent と共同で動作し、クラスターの MIB (Management Information Base) 用の計測を行います。

SNMP Master Agent と cmsnmpd は、クラスター関連のイベントについての通知 (トラップ) を行います。たとえば、クラスター構成の変更時や、Serviceguard パッケージの障害時に、トラップが送信されます。エージェントを構成して 1 つ以上の特定のあて先にトラップを送信するには、トラップのあて先を /etc/snmp/snmptrapd.conf に追加します (SUSE および Red Hat)。/etc/snmp/snmpd.conf の trap2sink でトラップがオンになっていることを確認してください (SUSE および Red Hat)。

cmsnmpd の rpm をインストールすると、snmpd と cmsnmpd が、自動的に起動するように構成されます。起動スクリプトは /etc/init.d/ にあります。このスクリプトを手動で起動して、デーモンの起動や停止を行うことができます。

詳細は、cmsnmpd (1) のマンページを参照してください。

クラスター WBEM エージェントデーモン: cmwbemd

このデーモンは、Serviceguard WBEM プロバイダープラグインモジュール (SGProvidersModule) および WBEM サービス cimserver と連携して動作し、cimserver にサブスクリプションを登録している Serviceguard WBEM Indication サブスクライバーに Serviceguard クラスターイベント (WBEM Indication) の通知を提供します。たとえば、クラスター構成の変更時や、Serviceguard パッケージの障害時に、Indication が送信されます。

cmwbemd を起動するにはコマンド /sbin/init.d/cmwbemagt start を、停止するにはコマンド /sbin/init.d/cmwbemagt stop を使用します。

プロキシデーモン: cmproxyd

このデーモンは、Serviceguard 構成データをプロキシまたはキャッシュして、ローカルノード上で実行されている特定の Serviceguard コマンドが構成データを使用できるようにするために使用されます。これにより、これらのコマンドがデータをすばやく取得できるようになり、cmcltd からの特定の要求に対する応答の負荷が軽減されます。

クラスターマネージャの動作

クラスターマネージャを使って、クラスタの初期化、クラスタの正常動作の監視、ノード障害の検出、ノードをクラスタへ追加/削除した際のクラスタの再編成の調整を行います。クラスターマネージャは、デーモンプロセスとして各ノード上で実行されます。クラスタの起動時や再編成時に、1つのノードがクラスタコーディネーターとして選択され、動作します。すべてのノードは何らかのクラスタ管理機能を実行していますが、クラスタコーディネーターはノード内通信の中心的役割を果たします。

クラスタの構成

システム管理者は、クラスタの構成パラメーターを設定して、クラスタの最初の起動を行います。その後、クラスタは通常の動作時に自動的に調整を行うので、人が介入して調整する必要はありません。クラスタの構成パラメーターには、クラスタの名前、ノード、クラスタのハートビートのネットワークパラメーター、クラスタのロック情報、タイミングパラメーターがあります(詳しくは、[第4章 \(72 ページ\)](#) を参照)。クラスタのパラメーターを入力するには、クラスタ構成ファイルを編集します(「[クラスタの構成](#)」(141 ページ) を参照)。どちらの方法を使っても、クラスタ構成ファイルをバイナリに変換し、クラスタ内のすべてのノードへ配布します。このファイルはクラスタ内のすべてのノードで同一でなければなりません。

ハートビートメッセージ

クラスターマネージャの主な作業は、クラスタ内のノード間のハートビートメッセージの送受信です。クラスタ内の各ノードは、ハートビートデバイスとして構成された各 IP ネットワークを使って、他のすべてのノードと UDP ハートビートメッセージを交換します。

1 台のクラスタノードが、規定時間内に他のすべてのクラスタノードからハートビートメッセージを受信しなかった場合、クラスタの再編成が開始されます。「[ノードがタイムアウトしたときの動作](#)」(68 ページ) を参照してください。新しいノードによってクラスタが編成されるようになった場合には、再編成の終了時にその情報がパッケージコーディネーターに渡されます(詳細は、この章の後述の「[パッケージマネージャの動作](#)」(38 ページ) を参照してください)。新しいクラスタに属さないノード上でこれまで実行されていたフェイルオーバーパッケージは、引き継ぎノードへ移されます。

ハートビートとデータが同じ LAN サブネットを通じて送信される場合、データ輻輳によって Serviceguard がハートビートを見逃し、データ輻輳がなければ不要であったクラスタの再編成を開始する場合があります。このため、ハートビート用に専用の LAN を使うとともに、データネットワーク上でハートビートを構成することをお勧めします。

各ノードは、Serviceguard が `MEMBER_TIMEOUT` パラメーターの値に基づいて計算した速度でハートビートメッセージを送信します。このパラメーターは、クラスタ構成の一部として作成するクラスタ構成ファイルで設定します。

- ❶ **重要:** 複数のハートビートが構成されていると、ハートビートは並行して送信されます。ノードが正しく稼働していることを確認するためには、Serviceguard は少なくとも 1 つのハートビートを受信しなければなりません。クラスタノードを相互接続するすべてのサブネットをハートビートネットワークとして構成することをお勧めします。これにより、コストを追加せずに複数の障害に対するデータ保護が強化されます。

ハートビート IP アドレスは、各ノード上で同じサブネットに属していることが必要ですが、複数のサブネットにまたがるようにクラスタを構成できます。「[クロスサブネット構成](#)」(25 ページ) を参照してください。ハートビートの要件についての詳細は、「[クラスタ構成のパラメーター](#)」(82 ページ)にある `HEARTBEAT_IP` を参照してください。タイムアウトの要件および推奨事項については、同じセクションにある `MEMBER_TIMEOUT` パラメーターの説明を参照してください。トラブルシューティング情報については、「[MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスタの再編成](#)」(244 ページ) を参照してください。「[クラスタデーモン: cmcld](#)」(30 ページ) も参照してください。

クラスター全体の手動による起動

手動での起動により、クラスター構成内の全ノードで1つのクラスターが形成されます。通常は、クラスター全体の保守やアップグレードの後、または再構成後に、初めてクラスターを起動するときに手動で起動します。

起動前に、クラスター内のすべてのノードに同じバイナリ形式クラスター構成ファイルがなければなりません。システム管理者は、いずれかのノードから `cmruncl` コマンドを実行することによりクラスターを起動します。`cmruncl` コマンドを使用できるのは、クラスターが実行されていない場合、つまりどのノードも `cmclld` デーモンを実行していない場合に限りです。

起動時に、クラスターマネージャソフトウェアは、起動コマンドで指定されているすべてのノードがクラスターの有効なメンバーかどうか、正常に起動して動作しているか、クラスターを形成しようとしているか、ノード間で互いにデータをやりとりできるか、などをチェックします。以上の項目を確認した後、クラスターマネージャはクラスターを形成します。

クラスターの自動起動

ノードがリブートしてクラスターへの結合が行われる場合は、いつでもクラスターが自動起動されます。このような状況は通常、各ノードのリブート後、または長期間の電源異常によりすべてのSPUが停止したなど、クラスター内のすべてのノードで障害が発生した場合に発生します。

クラスターの自動起動は、`$SGCONF/cmcluster.rc` ファイルのフラグ `AUTOSTART_CMCLD` が1に設定されている場合に実行されます。いずれかのノードでこのパラメーターを1に設定してリブートした場合、既存のクラスターに再結合するか、または既存のクラスターがない場合は、新しいクラスターを形成します。

動的クラスター再編成

動的再編成は、実行中のクラスターにノードが結合したり、またはクラスターからノードが分離する際に発生する、クラスターのメンバー構成に関する一時的な変更です。したがって、再編成は、変更がその後も継続して保たれる再構成とは別個のものです。クラスターの再編成は以下の状況で行われます (すべての状況が記述されているわけではありません)。

- アクティブノードでSPU障害またはネットワーク障害が検出された。
- アクティブでないノードがクラスターに結合しようとしているが、このノード上ですでにクラスターマネージャデーモンが起動されている。
- クラスター構成にノードが追加された。または、クラスター構成からノードが削除された。
- システム管理者がノードを停止した。
- パッケージ障害によりノードが停止した。
- サービス障害によりノードが停止した。
- ネットワークデータ通信量過多のため、クラスターがハートビート信号を受信できなかった。
- ハートビートネットワークに障害が発生したが、他のネットワークはハートビートを搬送するよう構成されていない。

通常のクラスターの再編成では、クラスター構成が変更されます。新しいクラスター構成では、以前の構成と比べてノード数が増減します。

スプリットブレインの問題を回避するクラスター定足数

一般に、クラスター再編成のアルゴリズムでは、クラスター定足数 (cluster quorum) としてこれまで稼働していたノードの過半数を必要とします。これまで稼働していたクラスターのちょうど半数のメンバーでクラスターが2つ再編成された場合、同じクラスターの2つのインスタンスが稼働するというスプリットブレイン状態になります。スプリットブレイン状態では、アプリケーションの複数インスタンスが同じディスクに同時にアクセスすることになります。

1つのインスタンスが回復処理を起動し、もう一方のインスタンスがディスクの状態を変更するような状況が発生します。Serviceguard の定足数の要件は、スプリットブレイン状態を回避するように設計されています。

クラスターロック

通常、クラスターを再編成するためには 50% より多くのクラスター定足数を必要としますが、**それまで稼働していたノードのちょうど半数で再編成する場合、残りの半数のノードで再編成が実行されないことが保証されれば**、ちょうど半数のノードで新しいクラスターとして再編成することができます。この保証は、タイブレーカーを使って、2つの同じサイズのノードグループのどちらかを選択することで可能になります。これにより1つのグループでクラスターを形成し、もう一方のグループはシャットダウンします。このタイブレーカーは、クラスターロックとも呼ばれます。クラスターロックは、ロック LUN またはクォーラムサーバーにより実現されます。クラスターロックは、2ノードのクラスターで必要です。

クラスターロックは、稼働中のクラスターで障害が発生し、Serviceguard が新たなクラスターの編成を試みている間にそのクラスターが同一サイズの2つのサブクラスターに分割された場合にのみ、タイブレーカーとして使用されます。各サブクラスターは、どちらもクラスターロックを取得しようとします。クラスターロックを取得したサブクラスターが新しいクラスターを形成するので、2つのサブクラスターが同時に稼働するのを防止できます。2つのサブクラスターのサイズが異なる場合は、ノードの過半数を持つサブクラスターが新しいクラスターを形成し、クラスターロックは使用されません。

2ノード クラスターの場合、クラスターロックを構成する必要があります。この2つのノードの間の通信が切断された場合、クラスターロックを取得したノードがクラスターを引き継ぎ、他方のノードは停止します (システムリセット)。クラスターロックがない場合には、クラスター内のどちらかのノードで障害が発生すると、もう一方のノードも停止し、これによりクラスター全体が停止します。クラスターロックを取得しようとしている最中にクラスターロックが失敗すると、クラスターは停止することに注意してください。

クラスターロックとしてのロック LUN の使用

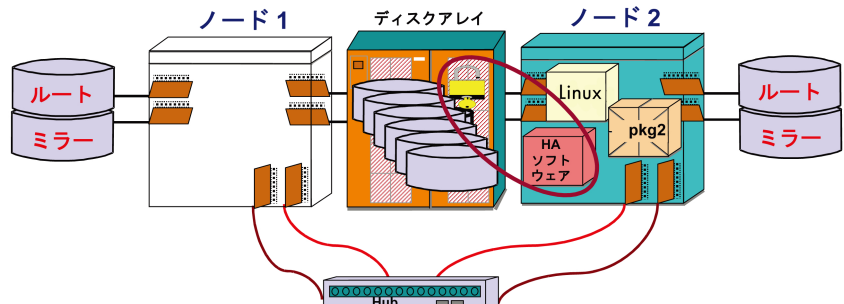
ロック LUN は、最大 4 ノードで構成されるクラスターで使用することができます。クラスターロック LUN は、クラスター内のすべてのノードで共有が可能なストレージの特別な部分 (パーティションといいます) です。あるノードがクラスターロックを獲得すると、「ロックが取得されている」ことを他のノードが認識できるように、このパーティションがマークされます。

注記: ロック LUN は、クラスターロック専用として使用します。また、ディスク全体をこの LUN に包含することをお勧めします。つまり、パーティションがディスク全体を占めるようにしてください。

ロック LUN の完全なパス名は、クラスター構成ファイルにリストされます。

ロック LUN の動作を図 7 に示します。

図 7 ロック LUN の動作



Serviceguard は定期的にロック LUN の状態をチェックし、ディスクの状態のチェックに失敗すると syslog ファイルにメッセージを書き込みます。ロックディスクの異常を早期に検出するためには、このファイルを監視する必要があります。

クラスターロックとしてのクォーラムサーバーの使用

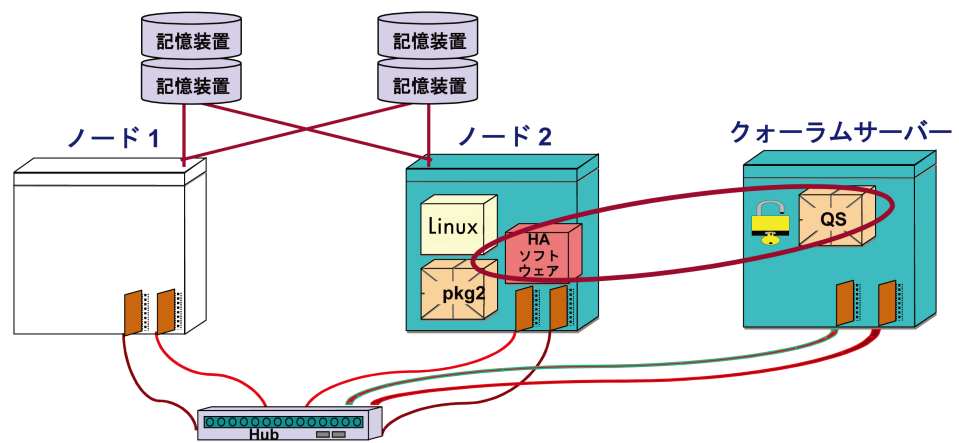
Linuxでのクラスターロックはクォーラムサーバーを使用して実装することもできます。クォーラムサーバーは、任意の規模のクラスターで使用できます。クォーラムサーバーソフトウェアは、Serviceguard パッケージとして構成するか、スタンドアロンで構成できます。いずれの場合も**クォーラムサービスを提供するクラスター以外**のシステムで実行する必要があります。

クォーラムサーバーは、Serviceguard ノードからの接続要求を既知のポートで受け付けます。このサーバーは、クラスターごとに特別なメモリ領域を用意します。1つのノードがクラスターロックを取得すると、「ロックが取得されている」ことを他のノードが認識できるように、この領域がマークされます。

クラスターの再編成中、タイブレークサービスが必要なときにクォーラムサーバーが利用できないと、クラスターは停止します。

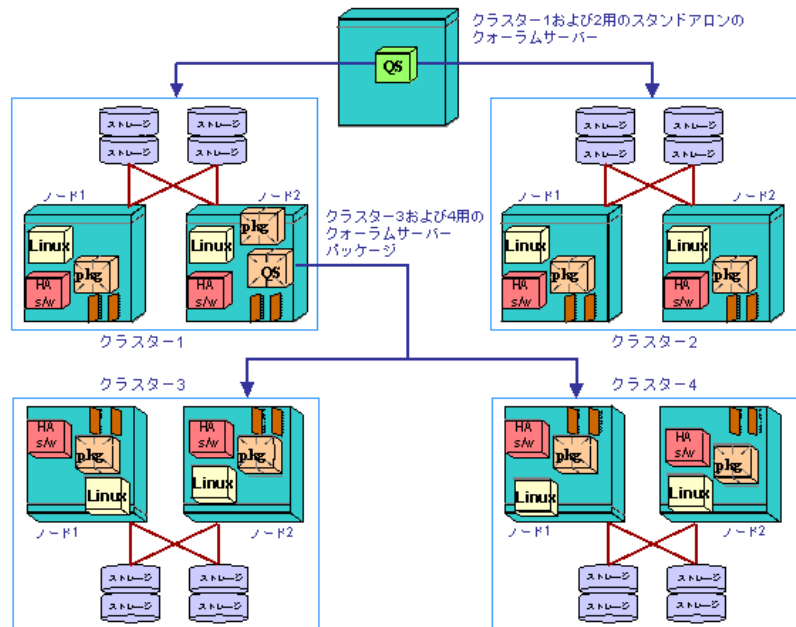
クォーラムサーバーの動作を、[図 8](#)に示します。ノード 1 とノード 2 の間の通信ができなくなった場合、クォーラムサーバーはノードを 1 つ (この例ではノード 2) 選択してクラスターの動作を続けます。他のノードは停止します。

図 8 クォーラムサーバーの動作



クォーラムサーバーは複数のクラスターにクォーラムサービスを提供できます。[図 9](#)に 4 つのクラスターでクォーラムサーバーを使用する例を示します。

図 9 クォーラムサーバーと複数クラスター



- ① **重要:** クォーラムサーバーについての詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にある『HP Serviceguard Quorum Server Release Notes』の最新版を参照してください。

クラスターロックなし

通常は、ノード数が3つ以下のクラスターをクラスターロックなしで構成しないでください。2 ノードクラスターではクラスターロックが必要です。3 ノード以上の構成には、クラスターロックを使用しないことも可能です。ただし、どのクラスターもタイブレークが必要となる可能性があることを考慮して決める必要があります。たとえば、保守作業のために3 ノードクラスターからノードを1つを除去すると、クラスターは2 ノードクラスターとして再編成されます。この後ノード障害または通信障害によりタイブレーカーを必要とする状況が発生すると、クラスター全体が使用できなくなります。

ノード数が5つ以上のクラスターでは、クラスターが同じサイズで二分割される可能性は小さいので、クラスターロックは必要ありません。ただし、ちょうど半数のノードで障害が同時に発生しないようにクラスターを構成してください。たとえば、同数のノード間で単一 LAN などの単一障害点が発生する可能性がないこと、また1つの電源系統にちょうど半数のノードを接続していないことを確かめてください。

オンラインでクォーラム構成を変更したときの動作

クラスターの稼働中にクォーラム構成を変更することができます。たとえば、クォーラム手法の変更 (ロックディスクからクォーラムサーバーへの変更など)、クォーラムデバイスの変更 (別のクォーラムサーバーへの変更など)、これらを制御するパラメーターの変更 (クォーラムサーバーのポーリング間隔の変更など) が行えます。クォーラムサーバーとロックパラメーターについての詳細は、「[クラスター構成のパラメーター](#)」(82 ページ) を参照してください。

クォーラム構成を変更すると、Serviceguard によって以下の2つの手順の処理が実行されます。

1. すべてのノードが厳格な過半数クォーラムに切り替えられる (既存のクォーラムデバイスがすべてオフになる)。

2. すべてのノードが新しく構成されたクォーラム方法、クォーラムデバイス、およびクォーラムパラメーターに切り替えられる。

- ① **重要:** 構成を変更する前にクラスターでクォーラムデバイスを使用していた場合、手順 1 の実行中に厳格な過半数クォーラムを使用しているノードで障害が発生すると、クラスターが突然停止する可能性があります。例として、2 ノードクラスターが稼働している間にクォーラムサーバーのポーリング間隔を変更する場合を考えてみましょう。手順 1 の実行中に 1 台のノードに障害が発生すると、クラスターがクォーラムを失って停止します。これは、元のクラスターメンバーの厳格な過半数 (この場合は 2 台) が要求されるためです。通常、手順 1 に要する時間は 1 秒前後であるため、ノードに障害が発生する可能性はごくわずかです。

時間間隔をできる限り短くするために、変更を適用する際にはクォーラム構成だけを変更し、他のものは変更しないようにしてください。

クラスター停止を引き起こす、このわずかなノード障害リスクも許容できない場合は、クォーラム構成を変更する前にクラスターを停止してください。

パッケージマネージャーの動作

パッケージとは、構成されたアプリケーションを Serviceguard が起動および停止する手段のことです。パッケージは、サービス、ディスクボリューム、汎用リソース、および IP アドレスの集合であり、これらは Serviceguard によって管理され、使用可能な状態が維持されます。

クラスター内の各ノードは、パッケージマネージャーのインスタンスを実行します。クラスターコーディネーター上に常駐するパッケージマネージャーを、パッケージコーディネーターと呼びます。

パッケージコーディネーターは、以下のことを行います。

- パッケージを実行、停止、移動する時期と場所の決定

パッケージマネージャーは、以下のことを行います。

- パッケージおよびサービスの実行、停止を行う制御スクリプトの実行
- 監視されているリソースの状態変化への対応

パッケージのタイプ

クラスターでは 3 種類のパッケージを実行できます。最も一般的なパッケージはフェイルオーバーパッケージです。同時に複数のノードで動作し、フェイルオーバーすることがない、特殊な目的のパッケージもあります。このようなパッケージは、通常、特定のフェイルオーバーパッケージのリソースを管理するために使われます。

フェイルオーバーしないパッケージ

フェイルオーバーすることがなく、同時に複数のノードで動作する、2 種類の特殊な目的のパッケージがあります。クラスター内のすべてのノードで動作するシステムマルチノードパッケージと、クラスター内のすべてまたは一部のノードで動作するように構成できる マルチノードパッケージです。システムマルチノードパッケージは、当社が提供するアプリケーション用に予約されています。

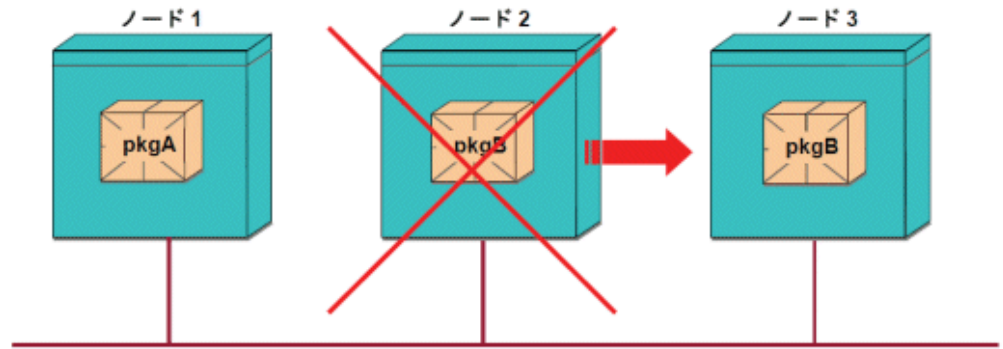
この項の残りの部分では、フェイルオーバーパッケージについて説明します。

フェイルオーバーパッケージ

フェイルオーバーパッケージは、クラスターの起動時に、適切なノード ([node_name\(164 ページ\)](#) を参照) 上で実行を開始します。サービスやネットワークなどのリソースの障害または依存関係の障害が発生した場合は、パッケージのフェイルオーバーが実行されます。パッケージのフェイルオーバーにより、既存のパッケージが停止するとともに、パッケージの新しいインスタンスが新しいノード上で起動します。

以下の図を参照してください。

図 10 フェイルオーバー中に移動するパッケージ



フェイルオーバーパッケージの構成

各パッケージは、個別に構成します。フェイルオーバーパッケージを作成するには、パッケージ構成ファイルテンプレートを生成して編集した後、パッケージをクラスター構成データベースに追加します。詳細は、第6章「パッケージとサービスの構成」(157ページ)を参照してください。

従来のパッケージ (A.11.18 より前のバージョンの Serviceguard で使われていた方法で作成されたパッケージ) では、パッケージサービスの実行を管理するパッケージ制御スクリプトも各パッケージに対して作成しなければなりません。詳細は、「従来のパッケージの構成」(218ページ)を参照してください。

カスタマイズされたパッケージ制御スクリプトは、モジュラーパッケージ (Serviceguard A.11.18 で導入された方法で作成されたパッケージ) では必要ありません。これらのパッケージは、Serviceguard とともにインストールされるマスター制御スクリプトにより管理されます。モジュラーパッケージの作成手順については、第6章「パッケージとサービスの構成」(157ページ)を参照してください。

フェイルオーバーパッケージを実行/停止する時期と場所の決定

パッケージ構成ファイルには、パッケージに名前が割り当てられており、パッケージが実行可能なノードのリストが記述されています。

フェイルオーバーパッケージでは、優先順位順にノードがリストされています (すなわち、リストの最初のノードの優先順位が最も高い)。さらに、フェイルオーバーパッケージのファイルには、フェイルオーバー動作を決定する次の3つのパラメーター `auto_run`、`failover_policy`、`failback_policy` が記述されています。

フェイルオーバーパッケージの切り替え動作

`auto_run` パラメーター (Serviceguard の初期のバージョンでは `PKG_SWITCHING_ENABLED` パラメーター) は、クラスター起動時のフェイルオーバーパッケージのデフォルトのグローバル切り替え属性、つまりクラスターの起動時にパッケージを自動的に起動させるかどうか、また障害が発生した場合パッケージを自動的に別のノードで再起動させるかどうかを定義します。各パッケージの切り替え属性は、クラスターの起動後に、`cmmmodpkg` コマンドで一時的に設定できます。リブートすると、構成した値が復元されます。

`auto_run` パラメーターは、パッケージ構成ファイルに設定されています。

通常のパッケージ切り替えでは、フェイルオーバーパッケージとそれに関連付けられた IP アドレスを新しいシステムに移動します。新しいシステムでは、すでに同じサブネットが適切に構成され、動作している必要があります。そうでなければ、パッケージは起動されません。

注記: ルーターで結合された複数のサブネットにまたがったクラスターを構成することができます。この構成では、一部のノードがあるサブネットを利用し、別のノードが別のサブネットを利用します。これを、クロスサブネット構成といいます。このような環境では、あるサブネット上のノードから別のサブネット上のノードにフェイルオーバーするようにパッケージを構成できるため、パッケージが開始するように構成されている各サブネットには、再配置可能 IP アドレスを構成する必要があります。「[クロスサブネットフェイルオーバーについて](#)」(118 ページ)、特に「[アプリケーションの配備への影響](#)」(119 ページ)を参照してください。

パッケージがフェイルオーバーすると、TCP 接続が切断されます。TCP アプリケーションは、再接続して接続を回復する必要があります。これは自動的に処理されません。パッケージが複数のサブネットに依存している場合、通常はパッケージを起動する前に、すべてのサブネットがターゲットノードで使用可能でなければならないので注意してください。(クロスサブネット構成では、パッケージに対して指定され、ターゲットノードに構成されているすべての監視対象サブネットが動作状態でなければなりません。)

パッケージがリソースや別のパッケージに依存している場合には、ターゲットノードでパッケージを起動する前に、依存関係が満たされている必要があります。

再配置可能 IP アドレスの切り替えを以下の図に示します。ユーザーは、使用するパッケージの IP アドレスを使って各ノードに接続します。各ノードはノードに関連する定常 IP アドレスを持ち、各パッケージはパッケージに関連する IP アドレスを持ちます。

図 11 パッケージの切り替え前

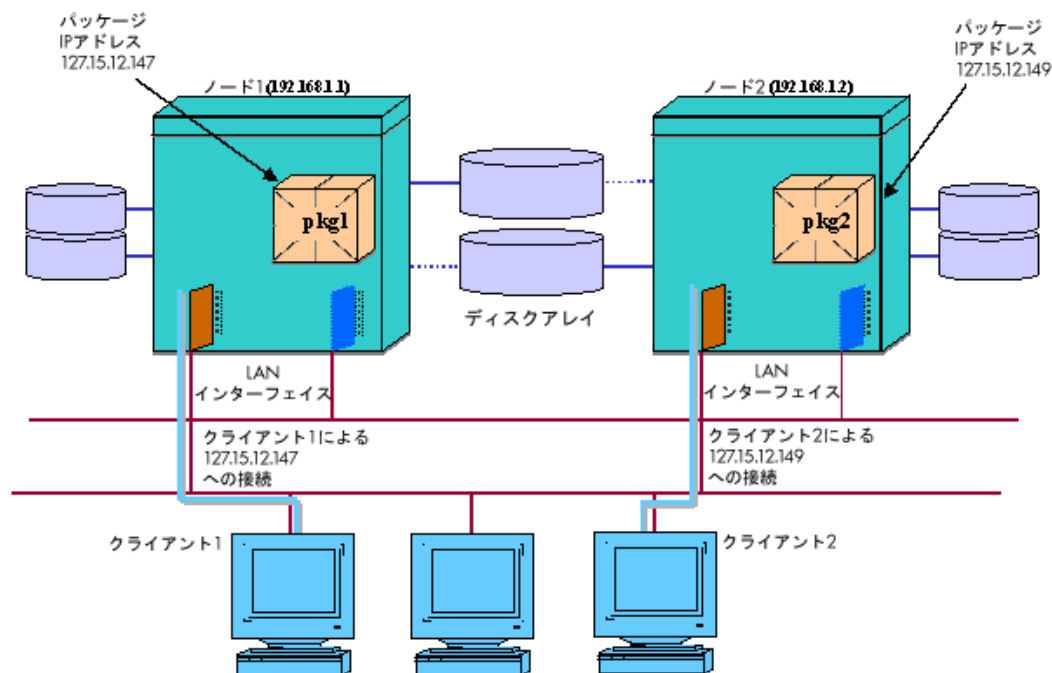
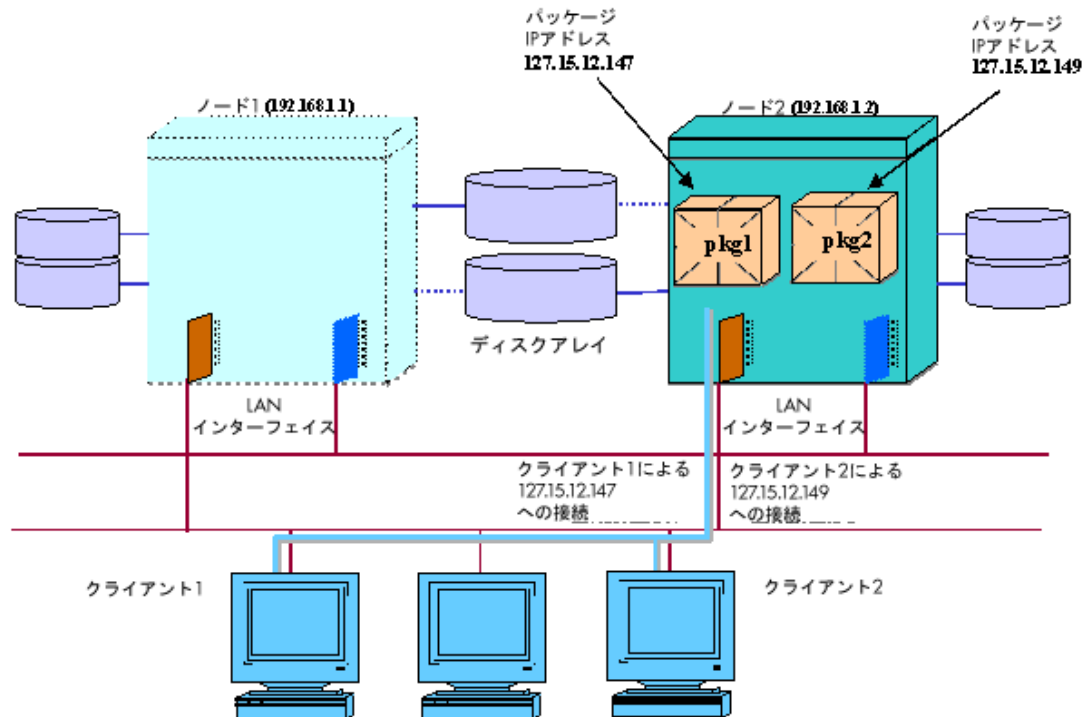


図 12では、ノード 1 に障害が発生し、pkg1 が ノード 2 に移動されています。pkg1 の IP アドレスは、パッケージとともに ノード 2 に移動しました。pkg1 は引き続き使用可能で、現在は ノード 2 で動作しています。また、ノード 2 は pkg1 のディスクと pkg2 のディスクのどちらにもアクセスできることにも注意してください。

注記: 複数のサブネットにまたがるクラスターの設計および構成については、「クロスサブネット構成」(25 ページ) に記載されている説明を参照してください。

図 12 パッケージの切り替え後



フェイルオーバーポリシー

パッケージマネージャーによるフェイルオーバーパッケージ実行ノードの選択は、パッケージ構成ファイルに記述されている優先順位リストと、同じくパッケージ構成ファイル中の *failover_policy* パラメーターに基づいています。フェイルオーバーポリシーにより、特定のノードが指定されていないときにパッケージを実行するノードや起動される必要のあるパッケージをパッケージマネージャーがどう選択するかを決定します。フェイルオーバーポリシーにより、フェイルオーバーの動作だけではなくパッケージの起動時の動作 (初回の起動も含めて) も決定されます。フェイルオーバー方針には、*configured_node* (デフォルト) と *min_package_node* の2つの方針があります。このパラメーターは、パッケージ構成ファイルに設定されています。

フェイルオーバー方針に *configured_node* を指定すると、パッケージは、ノードリスト中の使用可能なノードの内、優先順位が最も高いノード上で起動されます。フェイルオーバーが発生すると、パッケージは、ノードリスト中の使用可能なノードの内、次に優先順位の高いノードに移動します。

min_package_node をフェイルオーバー方針に指定すると、パッケージは、実行中の他のパッケージが最も少ないノードで起動されます。(ただし、負荷が最小のノードが選択されるとは限りません。ノード上で実行されているパッケージの数のみで決まります。)

自動交替用スタンバイ

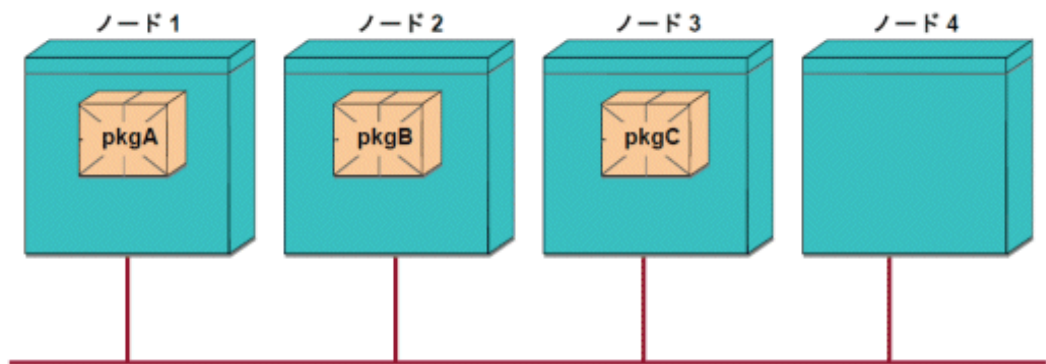
min_package_node フェイルオーバーポリシーを使用すると、1つのノードをクラスターに対する自動交代用スタンバイノードとして使用するクラスターを構成できます。以下に示すような4つのノードから成るクラスターを考えてみます。パッケージは、すべてどのノードでも実行可能で、同一の node_name リストを持っていると仮定します。ただし、例ではパッケージのノード名の順序が異なっていますが、これは必須ではありません。

表 2 パッケージ構成データ

パッケージ名	NODE_NAME リスト	FAILOVER_POLICY
pkgA	ノード 1、ノード 2、ノード 3、ノード 4	min_package_node
pkgB	ノード 2、ノード 3、ノード 4、ノード 1	min_package_node
pkgC	ノード 3、ノード 4、ノード 1、ノード 2	min_package_node

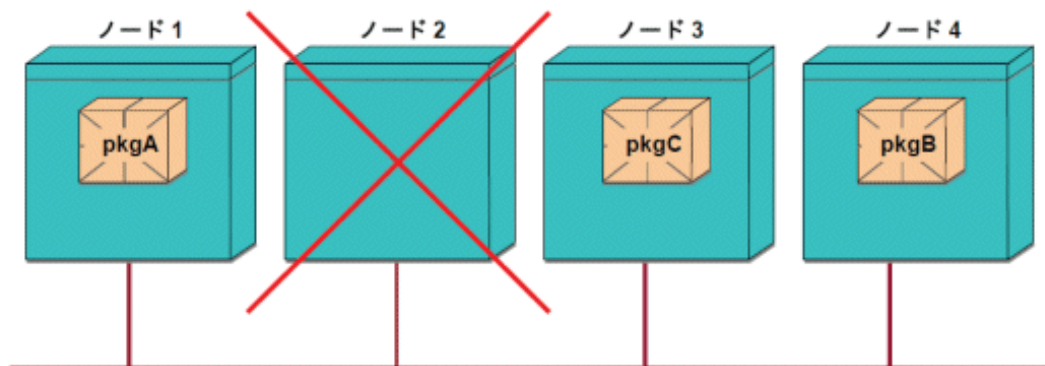
クラスターが起動すると、各パッケージは、図 13に示すように実行を開始します。

図 13 交替用スタンバイ構成 (フェイルオーバー前)



障害が発生すると、障害が起きたパッケージは、稼働中のパッケージが最も少ないノードにフェイルオーバーします。

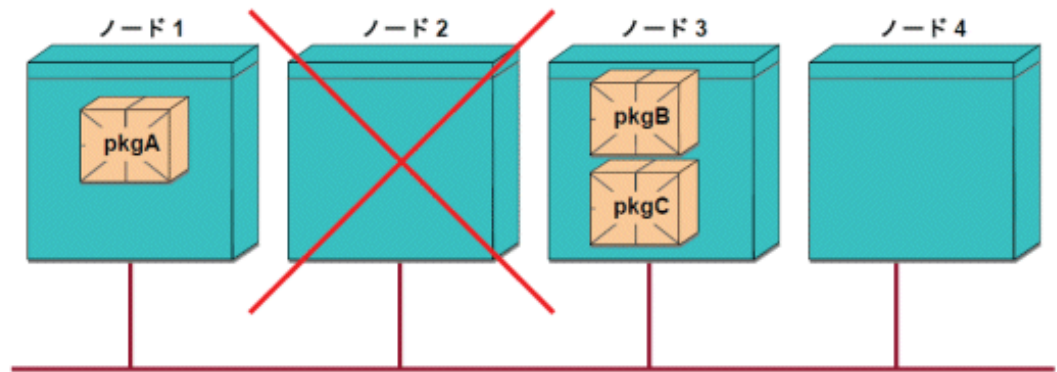
図 14 交替用スタンバイ構成 (フェイルオーバー後)



注記: `min_package_node` 方針では、ノード 2 を修理してクラスターに復帰させた時点で、ノード 2 が実行しているパッケージ数が最も少ないため、新しい待機ノードになります。

これらのパッケージが `configured_node` フェイルオーバー方針で設定されている場合、クラスター起動時は、図 13 のような状態になりますが、ノード 2 に障害が発生すると、パッケージは図 15 に示すように ノード 3 で実行されます。

図 15 `configured_node` 方針パッケージ (フェイルオーバー後)



フェイルオーバー方針として `configured_node` を使うと、パッケージは、ノードリスト中の使用可能なノードの内、優先順位が最も高いノード上で起動されます。フェイルオーバーが発生すると、パッケージは、ノードリスト中の使用可能なノードの内、次に優先順位の高いノードに移動します。

フェイルバックポリシー

`failback_policy` パラメーターを使用することにより、プライマリノードが利用可能になった時点で、プライマリノードからフェイルオーバーされていたパッケージをプライマリノードに戻すかどうかを決定することができます。プライマリノードとは、パッケージのノードリストの最上位にあるノードのことです。

このポリシーで考えられる 2 つの値は、`automatic` および `manual` です。このパラメーターは、パッケージ構成ファイルに設定されています。

例として、次の 4 ノード構成を示します。ここでは、`failover_policy` が `configured_node` に設定されており、`failback_policy` は `automatic` に設定されています。

図 16 自動フェイルバック構成 (フェイルオーバー前)

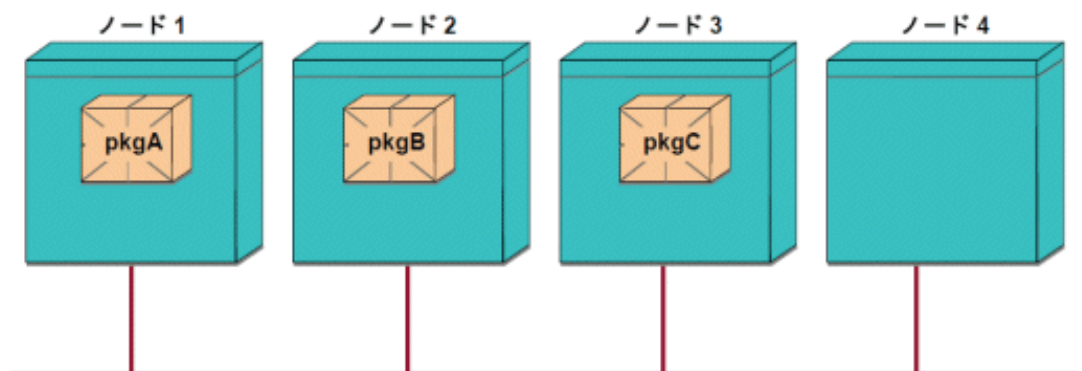
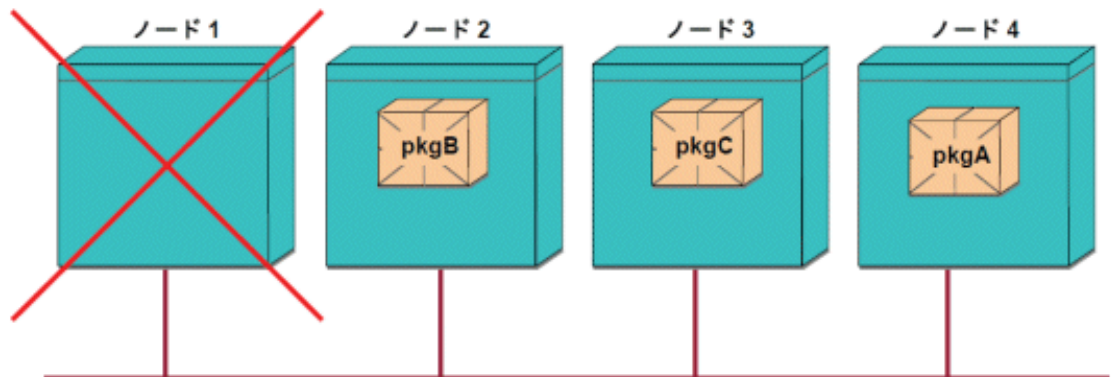


表 3 クラスターのノードリスト例

パッケージ名	NODE_NAME リスト	FAILOVER POLICY	FAILBACK POLICY
pkgA	ノード 1、ノード 4	configured_node	automatic
pkgB	ノード 2、ノード 4	configured_node	automatic
pkgC	ノード 3、ノード 4	configured_node	automatic

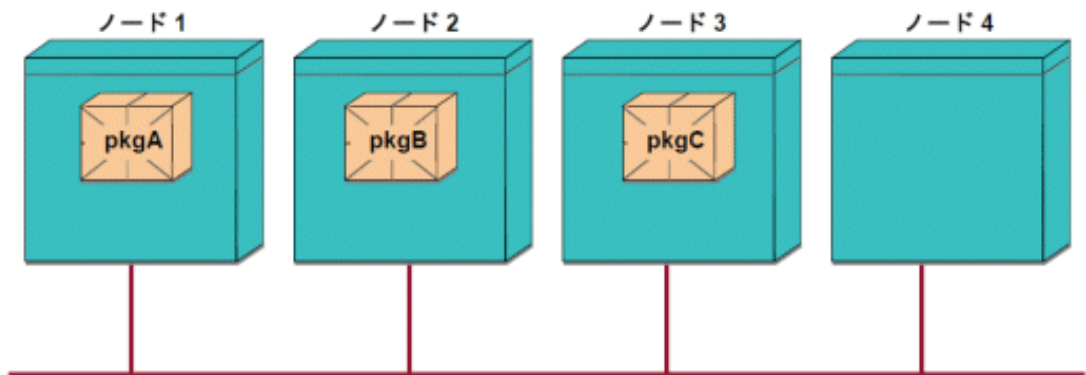
ノード 1 にシステムパニックが発生した場合、クラスターが再編成された後、pkgA は ノード 4 で実行されます。

図 17 自動フェイルバック構成 (フェイルオーバー後)



ノード 1 がリブートされ、クラスターに再結合された時点で、pkgA は自動的に ノード 4 上で停止され、ノード 1 上で再起動されます。

図 18 自動フェイルバック構成 (ノード 1 再起動後)



注記: `failback_policy` を `automatic` に設定した場合、業務上重要な時間帯にパッケージのフェイルバックが発生しアプリケーションが使用できなくなるおそれがあります。すなわち、パッケージを再びプライマリノードに切り替えている間、このパッケージによるサービスが一時的に使用できなくなります。したがって、自動フェイルバックを使用する場合は、パッケージのプライマリノードをクラスターに追加する時期を十分に検討してください。

`NODE_NAME` を「*」に設定すると、Serviceguard は自動的にパッケージのプライマリノードを選択します。`failback_policy` が `automatic` のときに `NODE_NAME` を「*」に設定し、クラスター内のノードを追加、削除したり、またはその名前を変更したりすると、パッケージのプライマリノードが変更されて、自動的にパッケージのフェイルオーバーが実行されることがあります。

フェイルオーバー方針とフェイルバック方針の組み合わせ

`failover_policy` の `min_package_node` と、`failback_policy` の `automatic` を組み合わせて使用すると、予測していなかったノード上でパッケージが実行される場合があります。これは、フェイルオーバーが発生したときに、パッケージの実行数が最も少ないノードが毎回同じノードであるとは限らないためです。

汎用リソース監視サービスの使用

汎用リソースモジュールは、パッケージのクリティカルリソースの監視を可能にする Serviceguard のリソース監視メカニズムです。このモジュールは、汎用リソースをパッケージ構成の一部として構成することで、Serviceguard でのカスタム、ユーザー定義モニターの統合を可能にします。汎用リソースを使用すると、カスタムモニターも含めて多様な監視メカニズムを使用でき、それらのメカニズムを 1 つのパッケージ内に共存させることができます。

汎用リソースには、次の利点があります。

- カスタム定義のモニターも統合できます。
- リソースのステータスの取得および設定では、制御が容易になり柔軟性が高まります。また、選択肢も増えます。

汎用リソースは、どのようなモジュラースタイルのパッケージにも構成できます。汎用リソースは、フェイルオーバーパッケージやマルチノードパッケージ用に構成できますが、デフォルトではモジュラーフェイルオーバーパッケージに含まれます。複数のパッケージに対して単一のリソースを指定できます。

汎用リソースモジュールパラメーターを含む新しいパッケージ構成ファイルを生成すること、既存のパッケージにモジュールを追加して汎用リソースパラメーターを含めることもできます。汎用リソースモジュールを使用してパッケージを生成する際、Serviceguard では、汎用リソースの構成のために次のパラメーターが提供されます。

- `generic_resource_name`
- `generic_resource_evaluation_type`
- `generic_resource_up_criteria`

これらのパラメーターを使用して汎用リソースを構成できます。パラメーターの詳細は、「[パッケージのパラメーターの説明](#)」(162 ページ) および `cmmakepkg (1m)` のマンページを参照してください。汎用リソースの構成手順については、「[汎用リソースの構成](#)」(97 ページ) を参照してください。

汎用リソースは、条件によっては、追加、削除、変更も可能です。詳細は、「[汎用リソースのオンライン再構成](#)」(100 ページ) を参照してください。

これらのリソースの監視は、Serviceguard 環境外部で行われます。これらの監視は、監視スクリプトを作成することで実行されます。監視スクリプトの起動は Serviceguard 環境内部からでも外部からでも行えますが、内部から行う場合は監視スクリプトをサービスとして構成する必要があります。

これらのスクリプトはエンドユーザーが作成しますが、リソースを監視し、それに合わせて `cmsetresource(1m)` を使って汎用リソースのステータスを設定するコアロジックを含む必要があります。スクリプトはパッケージの開始プロセスで起動され、パッケージサービスが停止するまで実行されます。詳細は、「[汎用リソース用の監視スクリプト](#)」(279 ページ)を参照してください。

複数のパッケージの一部として監視される必要のある共通の汎用リソースがある場合は、そのリソースの監視スクリプトを 1 つのパッケージの一部として起動できるため、他のすべてのパッケージが同じ監視スクリプトを使用できます。共通のリソースに対して複数のモニターを起動する必要はありません。監視スクリプトを起動したパッケージで障害が発生するかまたはパッケージが停止すると、この共通リソースを使っている他のすべてのパッケージも停止します。

「[監視スクリプトの起動](#)」(279 ページ) の、HP の推奨事項と例を参照してください。

汎用リソースは、シンプル汎用リソースと拡張汎用リソースの 2 つのタイプのいずれかに属します。

`up_criteria` パラメーターが指定されていない場合、その汎用リソースはシンプル汎用リソースと見なされます。

- シンプルリソースでは、監視メカニズムはリソースのステータスに基づきます。
- ステータスは、UP、DOWN、UNKNOWN のいずれかです。
- デフォルトステータスは、UNKNOWN で、UP および DOWN は、`cmsetresource(1m)` コマンドを使用して設定できます。

`up_criteria` パラメーターが指定されている場合、その汎用リソースは拡張汎用リソースと見なされます。

- 拡張リソースでは、監視メカニズムはリソースの現在の値に基づきます。
- 現在の値は、パッケージでリソースについて指定される `generic_resource_up_criteria` と比較され、汎用リソースのステータスが UP なのか DOWN なのかが決まります。
- デフォルトの現在値は 0 です。
- 有効な値は、1～2147483647 の正の整数です。

注記: シンプル/拡張汎用リソースのステータスと値の取得には `cmgetresource(1m)` コマンド、設定には `cmsetresource(1m)` コマンドを使用できます。詳細は、「[シンプル/拡張汎用リソースのステータスと値の取得と設定](#)」(99 ページ) およびマンページを参照してください。

1 つのパッケージでシンプルリソースと拡張リソースを組み合わせることもできますが、あるパッケージでシンプルリソースとして構成した汎用リソースを別のパッケージで拡張リソースとして構成することはできません。すべてのパッケージでシンプル汎用リソースまたは拡張汎用リソースのいずれかでなければなりません。

古いパッケージ構成ファイルの使用

旧バージョンの Serviceguard 用パッケージ構成ファイルを使用する場合は、`cmmakepkg` コマンドを使って新しいテンプレートを開き、その中にパラメーターの値をコピーすることをお勧めします。新しいテンプレート内の、元の構成を作成したときにはなかった選択肢の説明とデフォルト値に目を通してください。たとえば、`failover_policy` のデフォルトは、`configured_node` に変わり、`fallback_policy` のデフォルトは、`manual` に変わっています。

現在のパラメーターとそのデフォルト値についての詳細は、[第6章「パッケージとサービスの構成」](#) (157 ページ) と、パッケージ構成ファイルのテンプレートを参照してください。

パッケージの動作

パッケージとは、構成されたアプリケーションを Serviceguard が起動および停止する手段のことです。フェイルオーバーパッケージは Serviceguard のフェイルオーバー動作の単位でもあります。パッケージは、サービス、ディスクボリューム、汎用リソース、および IP アドレスの集合であり、これらは Serviceguard によって管理され、使用可能な状態が維持されます。1 クラスターあたり最大 300 パッケージ、つまり 1 クラスターあたり合計 900 のサービスと合計 100 の汎用リソースを使用することができます。

パッケージの実行

パッケージには 3 つの種類があります。

- フェイルオーバーパッケージは、最も一般的なパッケージです。同時には 1 台のノードでしか動作しません。障害が発生すると、構成ファイルに記述されている別のノードに移動します。移動できるノードが複数ある場合には、パッケージマネージャーはフェイルオーバーポリシーに従ってパッケージを起動するノードを決定します。
- システムマルチノードパッケージは、すべてのアクティブなクラスターノードで同時に動作します。このパッケージはすべてのノードで起動/停止し、個々のノードで起動/停止することはできません。
- マルチノードパッケージは、複数のノードで同時に動作します。Serviceguard は、`auto_run` に `yes` が設定されていると、構成ファイルに記述されているすべてのノードでマルチノードパッケージを起動します。このパッケージは、すべてのノードでも、個々のノードでも起動/停止することができます。停止するには、ユーザーコマンド (`cmhaltpkg`) を使います。また、サービス、サブネットなどのパッケージコンポーネントで障害が発生した場合には、Serviceguard が自動的に停止します。

システムマルチノードパッケージは、当社が提供するアプリケーションでのみ利用できます。フェイルオーバーパッケージは、マルチノードパッケージやシステムマルチノードパッケージに依存関係を持つように構成することができます。パッケージマネージャーは、パッケージが依存するパッケージがそのノードで起動され動作中でないと、パッケージを起動することができません。

パッケージマネージャーは、すべてのノードで実行を阻害する要因がある場合を除き、フェイルオーバーパッケージの実行の継続を試みます。フェイルオーバーパッケージが実行できない主な原因には、`auto_run` が無効になっているために Serviceguard がパッケージを起動できない、特定のノードでパッケージのノード切り替えが無効になっている、パッケージに設定されている依存関係が満たされていないなどがあります。あるノードで障害が発生したときに、パッケージの別のノードへの切り替えが有効になっていれば、依存関係が満たされている新しい場所で自動的に起動されます。このプロセスはパッケージ切り替えまたはリモート切り替えと呼びます。

フェイルオーバーパッケージは、構成ファイル内の最初の使用可能なノードで起動します。デフォルトでは、リスト内の次に使用可能なノードにフェイルオーバーします。障害が発生したフェイルオーバーパッケージを再起動するのに、必ずしも `cmrunpkg` コマンドを実行する必要はありません。多くの場合、`cmmodpkg` コマンドを使ってパッケージとノードの切り替えを有効にしておくのが最善の方法です。

パッケージを作成するときに、実行可能なノードのリストを指定します。システムマルチノードパッケージには、クラスター内のすべてのクラスターノードを指定する必要があります。マルチノードパッケージとフェイルオーバーパッケージには、一部またはすべてのクラスターノードを指定します。

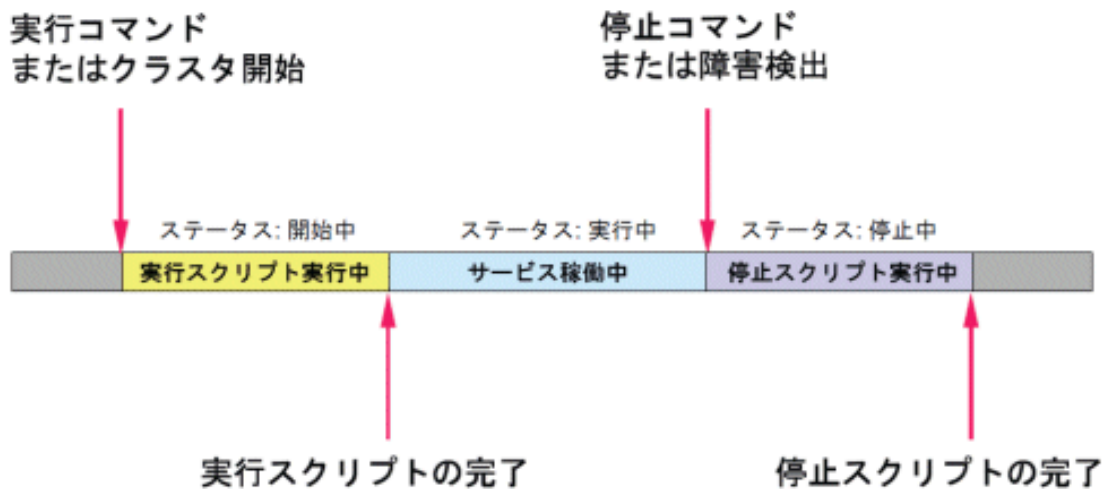
パッケージ構成ファイルの `auto_run` パラメーターに `yes` が設定されていると、クラスターの起動時にパッケージが Serviceguard によって自動的に起動されます。システムマルチノードパッケージでは、`auto_run` パラメーターに `yes` を設定しておく必要があります。フェイルオーバーパッケージで `auto_run` に `no` が設定されていると、クラスターの起動時にパッケージが Serviceguard によって自動的に起動されません。このような設定のパッケージでは、`cmmodpkg` コマンドを使って、明示的に有効にする必要があります。

注記: クラスターの実行中にパッケージを構成した場合、cmapplyconf コマンドが完了した後すぐにはパッケージが起動されません。クラスターを停止および再起動せずにパッケージを起動するには、cmrunpkg コマンドまたは cmmodpkg コマンドを実行する必要があります。

フェイルオーバーパッケージの起動、実行中の動作、パッケージのライフサイクルのフェーズの一部については、図 19 を参照してください。

注記: この図は、従来のパッケージにのみ適用されます。モジュラースクリプトでの相違点は、図の下の説明文に示します。

図 19 重要なイベントを示す従来のパッケージの時間表



パッケージのライフサイクルにおける最も重要な段階を以下に示します。

1. 制御スクリプトの起動前 (モジュラーパッケージの場合は、マスター制御スクリプト)
2. 実行スクリプトの実行中 (モジュラーパッケージの場合は、パッケージを起動する制御スクリプトの実行中)
3. サービスの実行中
4. 構成済みの汎用リソースがありその汎用リソースで障害が発生した場合、パッケージは停止します。
5. サービスまたはサブネットの異常終了時、または依存関係が満たされていない場合
6. 停止スクリプトの実行中 (モジュラーパッケージの場合は、パッケージを停止する制御スクリプトの実行中)
7. パッケージまたはノードのコマンドによる停止時
8. ノードの障害発生時

制御スクリプトの起動前

まず、ノードが選択されます。このノードは、パッケージのノードリストに記述され、パッケージのフェイルオーバーポリシーに一致し、パッケージに必要なリソースが、選択したノード上で使用可能でなければなりません。1つのリソースとして、パッケージに対して監視されるサブネットがあります。サブネットが使用できない場合、パッケージはこのノード上で起動できません。もう1つのリソースとしては、別のパッケージに対する依存関係があります。監視の結果、構成されているリソースに対する値が許容範囲内にはない場合には、パッケージは起動できません。

タイプが BPS の汎用リソースが構成されている場合はそれが稼働している必要があります。稼働していない場合、パッケージはこのノード上で起動できません。

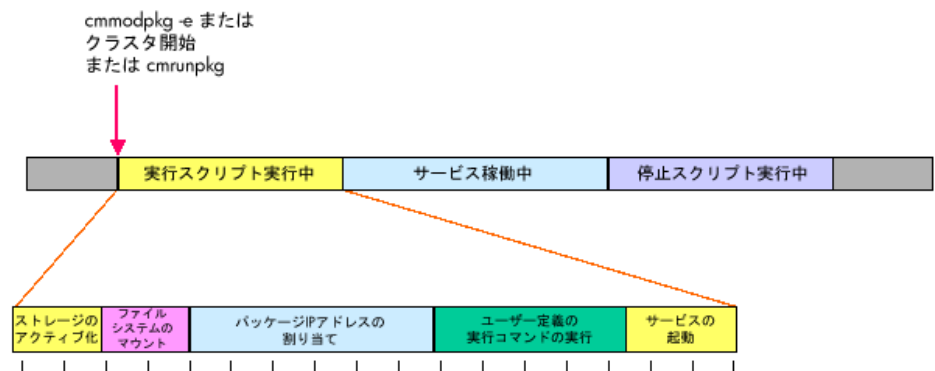
ノードが選択されると、そのノード上でパッケージを起動できるかどうかチェックが行われます。次に、選択したノード上で制御スクリプトにより、パッケージ用のサービスが起動されます。厳密には、選択したノード上で実行スクリプトを使って従来のパッケージを起動します。モジュラーパッケージは、マスター制御スクリプトが起動します。

実行スクリプトの実行中

パッケージマネージャーは、特定のノードでのパッケージの起動を決定すると、パッケージを起動するスクリプトを起動します (つまり、パッケージの制御スクリプトまたはマスター制御スクリプトが、`start` パラメーターを使って実行されます)。このスクリプトは、以下の手順を実行します。

1. `external_pre_script` があれば実行します (モジュラーパッケージのみ: 「外部スクリプトについて」 (115 ページ) を参照)。
2. ボリュームグループまたはディスクグループをアクティブにします。
3. ファイルシステムをマウントします。
4. ノードの LAN カードにパッケージの IP アドレスを割り当てます (フェイルオーバーパッケージのみ)。
5. ユーザー定義の実行コマンド (従来のパッケージのみ: 「パッケージ制御スクリプトへのユーザー定義関数の追加」 (222 ページ) を参照)、または `external_script` (モジュラーパッケージのみ: 「外部スクリプトについて」 (115 ページ) を参照) を実行します。
6. 各パッケージサービスを開始します。
7. 終了コード 0 で終了します。

図 20 従来のパッケージのパッケージの時間表



途中のいずれかの手順で、エラーが発生すると、スクリプトが異常終了します (終了コードは 1 です)。たとえば、パッケージのサービスを開始できなければ、制御スクリプトはエラーを発生して終了します。

注記: この図は、従来のパッケージに関するものです。モジュラーパッケージも、前述の通り、外部スクリプトと「プリスクリプト」を実行します。

`run_script_timeout` パラメーター(165 ページ)で指定した時間内に実行スクリプトの実行が終了しないと、パッケージマネージャーによりスクリプトは強制終了されます。実行スクリプトの実行中、ログファイルにメッセージが書き込まれます。従来のパッケージでは、このログファイルは実行スクリプトと同じディレクトリにあり、実行スクリプトと同じ名前で拡張子が `.log` です。モジュラーパッケージでは、パス名は、パッケージ構成ファイルの `script_log_file` パラメーター(166 ページ)で設定されます。通常の起動はログに記録され、パッケージの起動に関連するエラーメッセージや警告メッセージも記録されます。

注記: パッケージの実行スクリプトの作業が完了すると、そのスクリプトは終了します。つまり、パッケージが正常に起動されるとスクリプトは再び実行されません。スクリプトの終了後、スクリプトで起動されたサービスの PID は、パッケージマネージャーで直接監視されます。サービスが停止すると、パッケージマネージャーはパッケージの停止スクリプトを実行します。また `service_fail_fast_enabled`(172 ページ)が `yes` に設定されている場合は、パッケージマネージャーはパッケージが実行されているノードを停止します。サービスの再起動の回数がパッケージ制御スクリプトで指定されている場合、パッケージの実行スクリプトを再度実行することなく、再起動の最大回数までサービスが再起動されます。

実行スクリプトの正常終了および異常終了

実行スクリプトを終了する際の終了コードは、パッケージの次の動作を決定します。正常終了とはパッケージの起動が正しく行われたことを意味しますが、他のすべての終了は起動動作が正しく終了しなかったことを意味します。

- 0-正常終了。パッケージは正常に起動し、すべてのサービスがこのノードで実行されています。
- 1-異常終了。これは `no_restart` 終了とも呼びます。パッケージは、すべての起動手順を正常に終了しませんでした。サービスが強制終了され、パッケージは他のノードへフェイルオーバーすることができません。
- 2-代替終了。これは `restart` 終了とも呼びます。エラーが発生したが、パッケージは別のノードで起動することを許可されます。エラーが発生しても別のノードでパッケージを起動できる場合は、ユーザー定義のプロシーチャーで代替終了を使用することができます。 `restart` 終了のパッケージは、ローカルノードで実行することはできませんが、他のノードで実行することができます。
- タイムアウト-`run_script_timeout` で指定された時間が経過した場合に発生する、別のタイプの終了です。このような状況では、パッケージは強制終了され、全体で使用不能となります。ただし、現在使用しているノードでは使用不能となりません。パッケージスクリプトは、LVM ボリュームグループ、パッケージのマウントポイントなどのいくつかのリソースをクリーンアップできなかった可能性があります。したがって、ノードでパッケージを起動する前に、パッケージのリソースをクリーンアップする必要があるかどうか確認する必要があります。

`cmrunserv` を使ったサービスの起動

パッケージ制御スクリプトでは、`cmrunserv` コマンドにより個々のサービスが起動されます。このコマンドは、ファイル内に記述されている各サービスにつき 1 回実行されます。再起動回数は、各サービスに対して構成することができます。 `cmrunserv` コマンドはこの回数をパッケージマネージャーに渡し、サービスが異常終了した場合、サービスを該当数だけ再起動します。以下の設定は、従来のパッケージでの一般的な設定です。モジュラーパッケージでのサービスの構成についての詳細は、第 6 章の「 `service_name` 」(171 ページ)から始まる説明と、パッケージ構成テンプレートファイル内のコメントを参照してください。

```
SERVICE_RESTART[0]=" " ; do not restart
SERVICE_RESTART[0]="-r <n>" ; restart as many as <n> times
SERVICE_RESTART[0]="-R" ; restart indefinitely
```

注記: 再起動の `<n>` を設定し、`service_fail_fast_enabled` を `yes` に設定した場合、再起動の試行に `<n>` 回失敗した後にフェイルファーストが実行されます。サービスに対して `service_restart` を `"-R"` に設定し、`service_fail_fast_enabled` を `yes` に設定しても意味がありません。

サービスの実行中

クラスターサービスの正常動作中、パッケージマネージャーは以下のものを継続して監視します。

- サービスのプロセス ID
- パッケージ構成ファイル内で監視用に構成されたサブネット
- パッケージ構成ファイル内で監視用に構成された汎用リソース

サービスが異常終了しても、そのサービスの再起動パラメーターの値が 1 以上に設定されていれば、パッケージを停止することなく、設定した再起動回数までサービスが再起動されます。

正常動作の間すべてのサービスが実行中であれば、`cmviewcl` コマンドで出力される「スクリプトパラメーター」セクションで、サービスのステータスを参照することができます。

サービスやサブネットの異常終了時、または汎用リソースや依存関係が満たされない場合

障害が発生したらどうなるでしょうか? サービスが異常終了して再起動できない場合、あるいは別のパッケージに対する依存関係が満たされない場合は、使用中のノードでフェイルオーバーパッケージが停止し、パッケージ切り替えフラグの設定により、別のノードで再起動されます。マルチノードパッケージまたはシステムマルチノードパッケージで障害が発生すると、それに依存しているすべてのパッケージも障害状態になります。

パッケージの停止は、通常、パッケージ停止スクリプトが実行されることを意味します (次の項を参照してください)。ただし、フェイルオーバーパッケージの構成情報で、異常終了したサービスの `service_fail_fast_enabled` フラグ(172 ページ) が `yes` に設定されている場合、異常が検出されるとすぐにノードが停止します。このフラグが設定されていない場合、サービスの損失により、停止スクリプトが実行されてパッケージを停止します。

`auto_run`(164 ページ) が `yes` が設定されている場合は、起動の条件をすべて満たしていれば、別の使用可能なノードでパッケージが起動されます。`auto_run` を `no` に設定した場合は、パッケージを別のノードで起動することなく、そのパッケージが停止されます。

注記: パッケージがサブネットに依存しており、一次ノード上のサブネットに障害が発生すると、パッケージはシャットダウンを開始します。このサブネットがすぐに (パッケージが引き継ぎノード上で再起動される前に) 回復すると、パッケージマネージャーはこのパッケージを同じノード上で再起動します。パッケージ切り替えは発生しません。

コマンドによるパッケージの停止時

Serviceguard の `cmhaltpkg` コマンドは、パッケージ停止スクリプトの実行によって、特定のパッケージで実行中のサービスを停止します。これにより、パッケージのシャットダウンが行われ、パッケージの自動起動 (「`auto_run`」(164 ページ) を参照) を無効にします。

マルチノードパッケージまたはシステムマルチノードパッケージは、これらのパッケージに依存しているすべてのパッケージが停止していない限り、停止することはできません。`cmviewcl` を使って、依存しているパッケージのステータスを確認してください。たとえば、`pkg1` と `pkg2` が `PKGa` に依存している場合には、`PKGa` を停止する前に `pkg1` と `pkg2` の両方を停止する必要があります。

注記: `-n nodename>` オプションを指定して `<cmhaltpkg` コマンドを実行した場合、そのノードで実行されている場合のみ、パッケージが停止します。

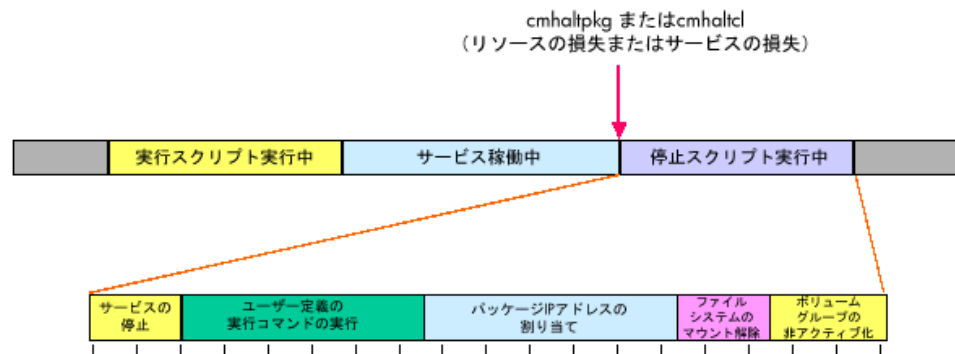
cmmodpkg コマンドは、パッケージの停止には使用できませんが、特定のノードまたはすべてのノードで切り替えを無効にすることはできます。切り替えを無効にしていると、パッケージの実行は継続できますが、現在実行中のノードで実行を停止しても、他のノードで再起動することはできません。

停止スクリプトの実行中

パッケージマネージャーが、フェイルオーバーパッケージが依存しているサービスまたはパッケージの異常を検出した場合や、特定のパッケージで cmhaltpkg コマンドが実行された場合、パッケージマネージャーは停止スクリプトを起動します。つまり、stop パラメーターを指定して、パッケージの制御スクリプトまたはマスター制御スクリプトが実行されます。このスクリプトは以下の手順を実行します (図 21 も参照してください)。

1. すべてのパッケージサービスを停止します。
2. ユーザー定義の停止コマンド (従来のパッケージのみ)、または `external_script` (モジュラーパッケージのみ: 「[external_script](#)」 (178 ページ) を参照) を実行します。
3. ノードの LAN カードからパッケージ IP アドレスを削除します。
4. ファイルシステムのマウントを解除します。
5. ボリュームグループを非アクティブ化します。
6. 永続的な登録および予約があれば、取り消します。
7. 終了コード 0 で終了します。
8. `external_pre_script` があれば実行します (モジュラーパッケージのみ: 「[external_pre_script](#)」 (178 ページ) を参照)。

図 21 停止スクリプトを実行するための従来のパッケージの時間表



途中のいずれかの手順で、エラーが発生すると、スクリプトが異常終了します (終了コードは 1 です)。halt_script_timeout (165 ページ) で指定した時間内に停止スクリプトの実行が終了しないと、パッケージマネージャーによりスクリプトは強制終了されます。停止スクリプトの実行中、ログファイルにメッセージが書き込まれます。従来のパッケージでは、このログ

ファイルは実行スクリプトと同じディレクトリにあり、実行スクリプトと同じ名前で拡張子が `.log` です。モジュラーパッケージでは、パス名は、パッケージ構成ファイルの `script_log_file` パラメーター(166 ページ) で設定されます。通常の起動はログに記録され、パッケージの停止に関連するエラーメッセージや警告メッセージも記録されます。

注記: この図は、従来のパッケージにのみ適用されます。モジュラスクリプトでの相違点は、図の上の説明文に示しています。

停止スクリプトの正常終了および異常終了

パッケージが他のノードへ移動できるかどうかは、停止スクリプトを終了する際の終了状況によって変わります。以下に、終了コードの候補値を示します。

- 0-正常終了。パッケージは正常に停止し、すべてのサービスがノード上でダウンしています。
- 1-異常終了。これは `no_restart` 終了とも呼びます。パッケージは正常には停止しませんでした。サービスが強制終了され、パッケージは全体で使用不能となります。ただし、現在使用しているノードでは使用不能となりません。
- タイムアウト-`halt_script_timeout` で指定された時間が経過した場合に発生する、別のタイプの終了です。このような状況では、パッケージは強制終了され、全体で使用不能となります。ただし、現在使用しているノードでは使用不能となりません。

パッケージ制御スクリプトのエラー条件および終了条件

表 4 に、フェイルオーバーパッケージのエラー条件、フェイルファースト設定およびパッケージ移動で発生する可能性がある組み合わせを示します。

表 4 フェイルオーバーパッケージのエラー条件およびパッケージ移動

パッケージエラー条件			結果			
エラーまたは終了コード	ノードフェイルファーストが使用可能か	サービスフェイルファーストが使用可能か	エラー後の一次ノード上の Linux ステータス	エラーまたは終了後の停止スクリプトの実行	エラー後プライマリノードでパッケージが実行可能か	代替ノードでパッケージが実行可能か
サービス障害	はい/いいえ	はい	システムリセット	いいえ	N/A (システムリセット)	はい
サービス障害	はい/いいえ	いいえ	実行中	はい	いいえ	はい
実行スクリプト (終了 1)	はい/いいえ	はい/いいえ	実行中	いいえ	変更なし	いいえ
実行スクリプト (終了 2)	はい	はい/いいえ	システムリセット	いいえ	N/A (システムリセット)	はい
実行スクリプト (終了 2)	いいえ	はい/いいえ	実行中	いいえ	いいえ	はい
実行スクリプト (タイムアウト)	はい	はい/いいえ	システムリセット	いいえ	N/A (システムリセット)	はい
実行スクリプト (タイムアウト)	いいえ	はい/いいえ	実行中	いいえ	変更なし	いいえ
停止スクリプト (終了 1)	はい	はい/いいえ	実行中	N/A	はい	いいえ
停止スクリプト (終了 1)	いいえ	はい/いいえ	実行中	N/A	はい	いいえ
停止スクリプト (タイムアウト)	はい	はい/いいえ	システムリセット	N/A	N/A (システムリセット)	はい。ただし <code>cmhaltpkg</code> コマンドの実行後

表 4 フェイルオーバーパッケージのエラー条件およびパッケージ移動 (続き)

パッケージエラー条件			結果			
エラーまたは終了コード	ノードフェイルファーストが使用可能か	サービスフェイルファーストが使用可能か	エラー後の一次ノード上のLinux ステータス	エラーまたは終了後の停止スクリプトの実行	エラー後プライマリノードでパッケージが実行可能か	代替ノードでパッケージが実行可能か
						にタイムアウトが発生しない場合。
停止スクリプト (タイムアウト)	いいえ	はい/いいえ	実行中	N/A	はい	いいえ
サービス障害	はい/いいえ	はい	システムリセット	いいえ	N/A (システムリセット)	はい
サービス障害	はい/いいえ	いいえ	実行中	はい	いいえ	はい
ネットワークの損失	はい	はい/いいえ	システムリセット	いいえ	N/A (システムリセット)	はい
ネットワークの損失	いいえ	はい/いいえ	実行中	はい	はい	はい
依存するパッケージの障害	はい/いいえ	はい/いいえ	実行中	はい	はい (依存関係が再び満たされた場合)	はい (依存関係が満たされている場合)

ネットワークマネージャーの動作

ネットワークマネージャーは、クライアントに対してネットワークサービスの高可用性を維持できるように、ネットワークカード障害を検出して、障害から回復することを目的としています。実際の動作としては、各パッケージのIPアドレスをパッケージが動作しているノード上のLANインターフェイスに割り当て、すべてのインターフェイスの状態を監視しながら、必要な場合にインターフェイスを切り替えます。

注記: Serviceguard は、ネットワークインターフェイス (NIC) の稼働状態を監視し、さらに IP レベル (レイヤー 3) ネットワークを監視できます。

定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット

各ノード (ホストシステム) には、アクティブなネットワークインターフェイスごとに 1 つの IP アドレスが必要です。このアドレスは定常 IP アドレスと呼ばれ、Red Hat では `/etc/sysconfig/network-scripts/ifcfg-<interface>` ファイル、SUSE では `/etc/sysconfig/network/ifcfg-<mac_address>` ファイルで構成されます。定常 IP アドレスはパッケージに対して与えられるものではないため、別のノードへ移動することはできません。

定常 IP アドレスは、データの送信、ハートビートメッセージの送信、またはその両方に使用します (「[クラスターマネージャーの動作](#)」(33 ページ) を参照)。これらのアドレスは、クラスター構成ファイルでクラスターに構成されます。「[クラスター構成のパラメーター](#)」(82 ページ) にある `HEARTBEAT_IP` および `STATIONARY_IP` のエントリーを参照してください。

Serviceguard は、これらの IP アドレスで示されるサブネットを監視します。これらのサブネットは、監視対象サブネットと呼ばれ、`cmviewcl` コマンドの出力でいつでもそのステータスを参照できます。例については、「[ネットワークのステータス](#)」(190 ページ) を参照してください。

パッケージ構成ファイル内の `monitored_subnet` パラメーター(169 ページ) を使って、サブネットをパッケージで監視するように構成することもできます。パッケージは、そのパッケージ構成ファイルの `monitored_subnet` で指定されたサブネットが稼働状態で、ノードから到達可能でなければ、起動されません。

- ① **重要:** パッケージ構成ファイルで `monitored_subnet` と指定したサブネットは、クラスター構成ファイル内の `NETWORK_INTERFACE` と `STATIONARY_IP` または `HEARTBEAT_IP` を使って、クラスターに構成する必要があります。「[クラスター構成のパラメーター](#)」(82 ページ) と「[パッケージのパラメーターの説明](#)」(162 ページ) を参照してください。

通常は、定常 IP アドレスの他に各パッケージに対して 1 つまたは複数の固有の IP アドレスを割り当てます。パッケージの IP アドレスは、パッケージの起動時に LAN インターフェイスに割り当てられます。

パッケージに対して与える IP アドレスは再配置可能 IP アドレス (または IP エイリアス、パッケージ IP アドレス、移動性 IP アドレス) と呼ばれます。これは、アドレスが実際に、あるクラスターノードから別のクラスターノードへ移動できるためです。再配置可能 IP アドレスは、1 つのクラスター内で最大 300 個のパッケージで合計 200 個まで使用できます。IPv4 アドレス、IPv6 アドレス、またはこの 2 つのアドレスファミリーを組み合わせたものを使用できます。システムマルチノードパッケージとマルチノードパッケージはフェイルオーバーしないため、再配置可能 IP アドレスは設定されません。

再配置可能 IP アドレスは、パッケージに割り当てられる仮想ホスト IP アドレスに似ています。各パッケージの名前は DNS (ドメインネームシステム) を使って構成することをお勧めします。これにより、プログラムはパッケージの名前をホスト名のように使用できるので、`gethostbyname(3)` への入力パラメーターとして使用して、パッケージの再配置可能 IP アドレスを取得することもできます。

パッケージの制御が移行された場合は、引き継ぎノードが再配置可能アドレスを引き継ぎます (定常アドレスは引き継がれません)。つまり、アプリケーションは、パッケージが現在どのノードにあるかを知らなくても、その再配置可能アドレスを使ってパッケージにアクセスできます。

- ① **重要:** パッケージで再配置可能アドレスに使用されるサブネットは、クラスター構成ファイルの `NETWORK_INTERFACE` と、`STATIONARY_IP` または `HEARTBEAT_IP` のいずれかを使ってクラスターに構成する必要があります。これらのパラメーターについての詳細は、「[クラスター構成のパラメーター](#)」(82 ページ) を参照してください。再配置可能アドレスの構成についての詳細は、パッケージの `ip_` パラメーター([170 ページ](#)) の説明を参照してください。

注記: ルーターで結合された複数のサブネットにまたがったクラスターを構成することができます。この構成では、一部のノードがあるサブネットを利用し、別のノードが別のサブネットを利用します。これを、クロスサブネット構成といいます。このような環境では、あるサブネット上のノードから別のサブネット上のノードにフェイルオーバーするようにパッケージを構成できるため、パッケージが開始するように構成されている各サブネットには、再配置可能アドレスを構成する必要があります。「[クロスサブネットフェイルオーバーについて](#)」(118 ページ)、特に「[アプリケーションの配備への影響](#)」(119 ページ) を参照してください。

IP アドレスの種類

Serviceguard では IPv4 アドレスと IPv6 アドレスの両方がサポートされます。IPv4 アドレスは、`n.n.n.n` 形式の従来からあるアドレスで、`n` は、0~255 の 10 進数です。IPv6 アドレスは、`x:x:x:x:x:x:x:x` 形式のアドレスで、`x` は 128 ビットのアドレスを 16 ビットごとに区切ってできる 8 つの部分それぞれを 16 進数で表わした値です。ハートビート IP、定常 IP、および再配置可能 (パッケージ) IP は、IPv4 アドレスまたは IPv6 アドレス (あるいはこの両方の組み合わせ) として定義できます。

再配置可能 IP アドレスの追加と削除

パッケージが起動されると、そのパッケージに構成された再配置可能 IP アドレスがすべて指定された IP サブネットに追加されます。パッケージが停止されると、再配置可能 IP アドレスは IP サブネットから削除されます。これらの機能は、**パッケージマスター制御スクリプト** (従来のパッケージではパッケージ制御スクリプト) 内の `cmmodnet` コマンドで実行されます。

IP アドレスが構成されるのは、各一次ネットワークインターフェイスカード上だけです。同じネットワークカードに複数の IPv4 アドレスを構成する場合は、同じ IP サブネットに属していなければなりません。

△ **注意:** パッケージへの再配置可能アドレスの追加は、パッケージ構成ファイル内の `ip_address` (171 ページ) (または従来のパッケージの制御スクリプト内の `IP []` エントリー) を編集し、`cmaplyconf (1m)` を実行するという方法で **のみ** 行うことを強くお勧めします。

負荷の分散

Serviceguard では、1 つの IP アドレスを共有する複数のサービスを 1 つのパッケージに構成することができます。この場合、パッケージがフェイルオーバーすると、これらのサービスがすべてフェイルオーバーします。サービスの負荷を分散させる (必要に応じて負荷の小さいシステムに特定のサービスを移動する) 場合は、各サービスをそれぞれのパッケージに置き、一意の IP アドレスを各サービスに付与します。

LAN インターフェイスのボンディング

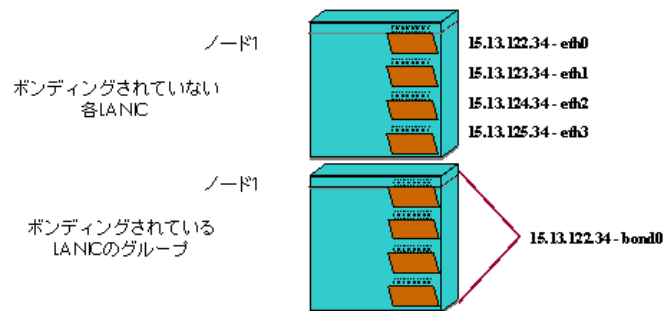
Linux のチャンネルボンディングと呼ばれるプロセスを利用すると、ノード上の複数の LAN インターフェイスを 1 つのグループにまとめることができます。ボンディンググループでは、一般的に 1 つのインターフェイスがデータの送受信に使われ、他のインターフェイスはバックアップとして利用されます。1 つのインターフェイスで障害が発生すると、ボンディンググループ内の他のインターフェイスが引き継ぎます。ネットワークサービスの可用性を高めるために、すべての主要な IP サブネットでチャンネルボンディングを使うことを強くお勧めします。

ホストバスアダプター (HBA) は同一である必要はありません。イーサネット LAN は同じタイプでなければなりませんが、バンド幅は異なっても構いません (たとえば、1 Gb と 100 Mb)。Serviceguard for Linux は、ドライバーレベルでの LAN インターフェイスのボンディングの使用をサポートしています。イーサネットドライバーは、インターフェイスのグループを使うように構成されています。

ボンディングが有効になると、各インターフェイスは、IP アドレスと MAC アドレスが 1 つだけある、複数の物理ポートを備えた論理リンク 1 つとして見えます。ボンド 1 つ当たりのスレーブ (ポート) 数には制限はありません。システム当たりのボンド数は、ロード可能な Linux モジュールの数によって制限されます。

ボンディングでは、マルチポートのネットワークカード (現在、ポート数が 4 つまでのものが入手可能) のポートをまとめることができます。また、個別のカードのポートに対してボンディングを適用することもできます。HP では個別のカードを使用することをお勧めします。図 22 に、4 つの個別のインターフェイスを 1 つのボンドにまとめる例を示します。

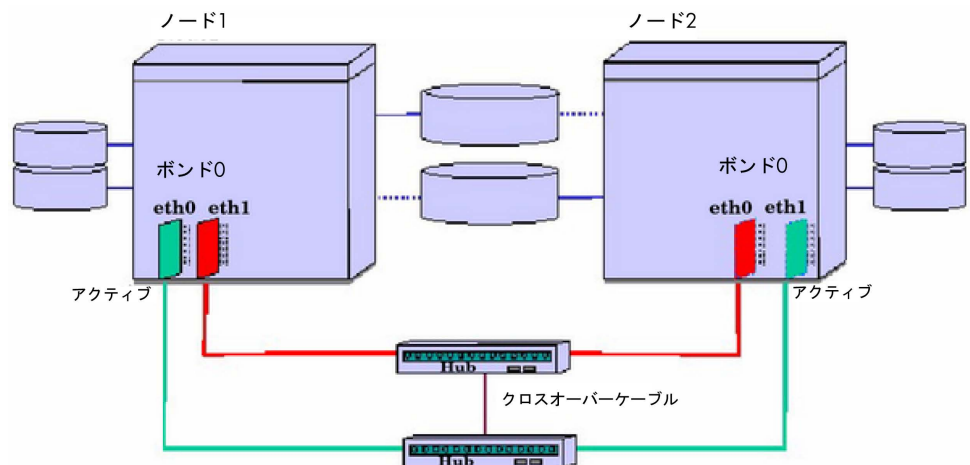
図 22 ボンディングされたネットワークインターフェイス



ボンディングされていない構成の LAN に 4 枚の LAN カードがあります。4 枚の LAN カードには、それぞれ独立した (集結されていない) IP アドレスと MAC アドレスが対応付けられ、それぞれ独自の LAN 名 (eth1、eth2、eth3、eth4) を持ちます。これらのポートが集結されている場合は、4 つのポート全体に対して 1 つの IP アドレスと MAC アドレスが対応付けられます。この例では、集結されたポートには bond0 という名前が付けられ、クラスターの構成の際には、この名前でボンドが認識されます。

図 3-18 に、クロスオーバーケーブル付きの冗長ハブを使った、ボンディング構成を示します。

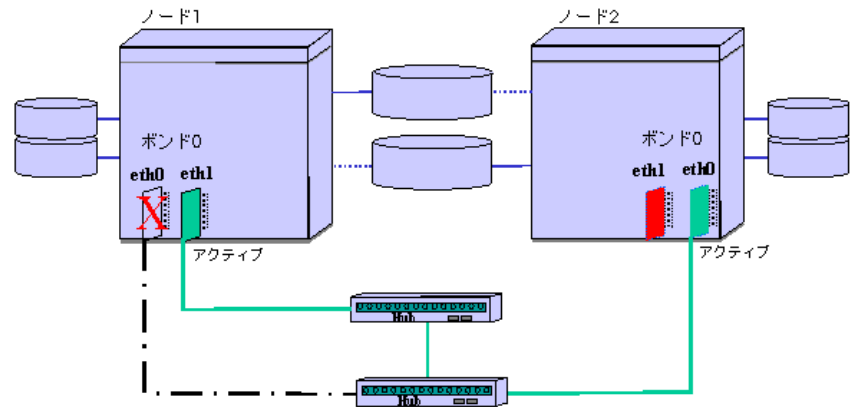
図 23 ボンディング NIC



ボンディングモデルでは、個々のイーサネットインターフェイスがスレーブで、ボンドがマスターになります。基本的な高可用性構成 (mode 1) では、ボンド内の 1 つのスレーブがアクティブで、他のスレーブは障害が検出されるまで非アクティブになります。(図 3-18 では、両方の eth0 インターフェイスがアクティブです。) 構成の際には、すべてのノード上のアクティブなスレーブインターフェイスを、同じハブに接続することが重要です。同じハブに接続しないと、LAN の通常動作でハブ間のクロスオーバーを使う必要が発生し、このクロスオーバーが単一障害点になる可能性があります。

カードで障害が発生した後も、メッセージはボンディングされた LAN 上で転送され続け、他のノードで受信されますが、ノード 1 の bond0 では eth1 がアクティブになっています。この状況を、図 24 に示します。

図 24 障害後のボンディング NIC



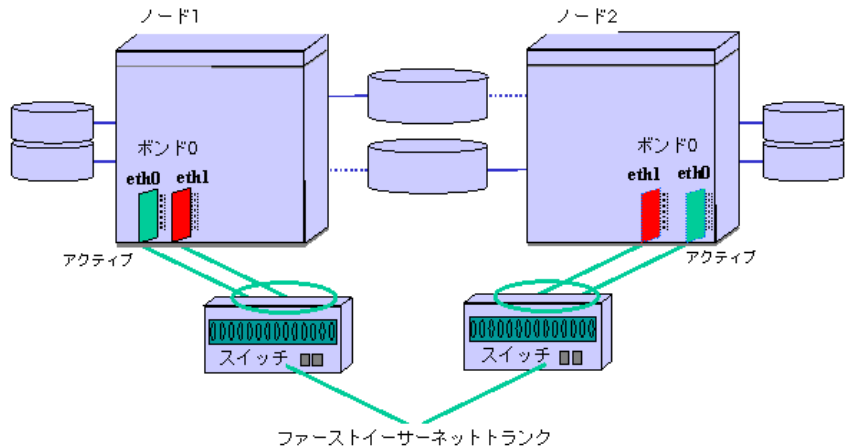
イーサネットカードの種類(シングルポートまたはデュアルポート)およびボンディンググループの組み合わせは複数考えられますが、どのような組み合わせのチャンネルボンディングでも、ハートビート接続の単一障害点を防ぐために、最低限 2 枚の物理カード (または、物理的に分離されているオンボード LAN インターフェイス) を使用することが重要です。

負荷バランスを目的としたボンディング

ボンディングを負荷バランスモードで構成し、すべてのスレーブをアクティブにしてデータを並列に送信することもできます。この場合、構成要素の LAN の 1 つに障害が発生してもボンディングが機能し続けるため、高可用性が確保されます。個々のハードウェア構成で、HP Procurve スイッチのような、スイッチポートのトラッキングをサポートするイーサネットスイッチを使用する必要があるかどうかについては、ボンディングのドキュメントで確認してください。ボンディングドライバーの構成ではボンディングタイプとして mode 0 を指定しなければなりません。

このタイプの構成を図 25 に示します。

図 25 負荷バランスを目的として構成されたボンディング NIC



LAN インターフェイスの監視と障害の検出: リンクレベル

Serviceguard は、`NETWORK_POLLING_INTERVAL` (「[クラスター構成のパラメーター](#)」(82 ページ)を参照)で定められた通常の間隔で、クラスター構成ファイル内で指定されたすべてのネットワークインターフェイスカード (ボンディングされているものとされていないもの両方) をポーリングします。インターフェイスのリンクステータスがダウンの場合、Serviceguard はそのインターフェイスと、その上で稼働しているすべてのサブネットを「down」とマークします。これは `cmviewc1 (1m)` の出力で確認できます (「[リンクレベルおよび IP レベルの障害の報告](#)」(62 ページ)を参照)。リンクが回復すると、Serviceguard はそのインターフェイスと、その上で稼働しているすべてのサブネットを「up」とマークします。

LAN インターフェイスの監視と障害の検出: IP レベル

Serviceguard では、IPv4 および IPv6 サブネットのレイヤ 3 の稼働状態と接続状況をチェックすることによって IP レベルを監視できます。これを行うには、構成可能な IP モニターを使用します。IP モニターは、クラスターに構成されている任意のサブネットに対して有効にすることができますが、常に監視する必要はありません。クラスターの稼働中にサブネットに対して IP モニターを構成したり、監視を無効にしたりできます。

IP モニター:

- ネットワークインターフェイスが IP メッセージの送受信に失敗したことを検知する。この検知は、リンクレベルでまだ稼働中であっても行われます。
- 障害、フェイルオーバー、回復、およびフェイルバックを処理します。

IP モニターを使用する理由

IP モニターでは、リンクレベルの監視で既に提供されている機能に加えて、以下のような機能が提供されます。

- 第一レベルのスイッチの先のネットワーク状態を監視できます。「[IP モニターの動作](#)」(60 ページ)を参照してください。
- 以下のようなエラーを検出して処理できます。
 - ルーターまたはスイッチでの IP パケットの破損

- スイッチと第一レベルルーターの間のリンク障害
- 受信障害
- パケットの受信を妨げるが、インターフェイスのリンクレベルの稼働状態には影響しないエラー

① **重要:** IP モニターはクロスサブネット構成で構成する必要があります。IP モニターでは、リンクレベルの監視で検出されないエラーが検出されるためです。「[クロスサブネット構成](#)」(25 ページ) も参照してください。

IP モニターの動作

IP モニターは、インターネット制御メッセージプロトコル (ICMP) と ICMPv6 を使って、ターゲットの IP アドレスにポーリングメッセージを送信し、応答を受信したことを確認します。IP モニターによって障害が検出されると、IP レベルでそのネットワークインターフェイスが「down」とマークされます。これは `cmviewcl (1m)` の出力で確認できます(「[リンクレベルおよび IP レベルの障害の報告](#)」(62 ページ) および「[障害および回復の検出時間](#)」(61 ページ) を参照)。

IP モニターは、以下の 2 種類のポーリングを実行できます。

- ピアポーリング。
この場合、IP モニターは、サブネット上の各 IP アドレスから、クラスター内の他のノードの同じサブネット上にある他のすべての IP アドレスに ICMP ECHO メッセージを送信します。
- ターゲットポーリング。
この場合、IP モニターは、サブネット上の各 IP アドレスから、クラスター構成ファイルで指定された外部 IP アドレスに ICMP ECHO メッセージを送信します。「[クラスター構成のパラメーター](#)」(82 ページ) の `POLLING_TARGET` を参照してください。`cmquerycl (1m)` は、以下の例に示すように、ポーリングターゲットとして使用できるゲートウェイを検出します。
ターゲットポーリングでは、第一レベルのスイッチの先を監視できます。これにより、各監視対象の IP アドレスとターゲットの間の任意の位置にある経路の障害を検出することができます。

注記: クロスサブネット構成では、他の経路指定されたサブネットのノード上のピアインターフェイスをポーリングターゲットとしてノードに構成できます。

ターゲットポーリングは、サブネットがクラスターにとってプライベートでない場合に構成することをお勧めします。

`cmquerycl` 出力の IP Monitor セクションは以下のようになります。

```
...
Route Connectivity (no probing was performed):

IPv4:

1    16.89.143.192
    16.89.120.0

...

Possible IP Monitor Subnets:

IPv4:

16.89.112.0          Polling Target 16.89.112.1
```

IPv6:

```
3ffe:1000:0:a801:: Polling Target 3ffe:1000:0:a801::254
```

...

IP 監視がターゲットポーリングで構成されているサブネットの場合、クラスター構成ファイルの IP Monitor セクションは以下ようになります。

注記: cmquerycl が対象のサブネットのゲートウェイを検出する場合、これがデフォルトです。詳細は、「[クラスター構成のパラメーター](#)」(82 ページ) の *SUBNET* を参照してください。

- ❗ **重要:** デフォルトでは、cmquerycl は、検出したゲートウェイが監視用に正常に動作するかどうかは検証しません。ただし、`-w full` オプションを使用すると、cmquerycl はゲートウェイをポーリングターゲットとして検証します。
-

```
SUBNET 192.168.1.0
  IP_MONITOR ON
  POLLING_TARGET 192.168.1.254
```

ピアポーリングを使ってサブネットに IP 監視を構成するには、クラスター構成ファイルの IP Monitor セクションを以下のように編集します。

```
SUBNET 192.168.2.0
  IP_MONITOR ON
```

注記: これはデフォルトではありません。cmquerycl は、対象のサブネットのゲートウェイを検出しない場合、*IP_MONITOR* を OFF に設定して、このサブネットの IP レベルのポーリングを無効にします。ゲートウェイを検出する場合は *POLLING_TARGET* を設定して、ターゲットポーリングを有効にします。詳細は、「[クラスター構成のパラメーター](#)」(82 ページ) の *SUBNET* を参照してください。

IP 監視が無効になっているサブネットの場合、クラスター構成ファイルの IP Monitor セクションは以下ようになります。

```
SUBNET 192.168.3.0
  IP_MONITOR OFF
```

注記: cmquerycl が対象のサブネットのゲートウェイを検出しない場合、これがデフォルトです。これはサブネットの *SUBNET* エントリーがない場合と同じです。詳細は、「[クラスター構成のパラメーター](#)」(82 ページ) の *SUBNET* を参照してください。

障害および回復の検出時間

IP モニターは、*NETWORK_POLLING_INTERVAL* (「[クラスター構成のパラメーター](#)」(82 ページ) を参照) をデフォルトの 2 秒で使った場合、通常、イーサネットでは 8~10 秒以内、InfiniBand では 16~18 秒以内に IP 障害を検出します。同様に、*NETWORK_POLLING_INTERVAL* をデフォルトで使った場合、通常、イーサネットでは 8~10 秒以内、InfiniBand では 16~18 秒以内に IP アドレスの回復を検出します。

IP アドレスの障害または回復の検出の最小時間は、イーサネットでは 8 秒、InfiniBand では 15 秒です。

- ❗ **重要:** *NETWORK_POLLING_INTERVAL* の値をデフォルトの 2 秒以外に変更しないことを強くお勧めします。
-

「[リンクレベルおよび IP レベルの障害の報告](#)」(62 ページ) も参照してください。

制約および制限事項

- サブネットを監視するには、そのサブネットがクラスターに構成されている必要があります。
- ポーリングターゲットの検出は、第1レベルのルーターを越えて行われることはありません。
- ポーリングターゲットは ICMP(または ICMPv6) の ECHO メッセージを受け取って応答する必要があります。
- ノードは常に自分自身に ping を実行できるため、同じサブネット上のピア IP をポーリングターゲットにすることはできません。

サブネット上にインターフェイスが2つしかない場合は、以下の制約がピアポーリングに適用されます。

- 一方のインターフェイスで障害が発生すると、ボンディングが構成されていて、かつ稼働中のスタンバイが存在しない限り、各ノードで両方のインターフェイスとサブネット全体が「down」とマークされます。
- 一方のインターフェイスを持つノードがダウンした場合、もう一方のノードのサブネットもダウンしているとみなされます。
- 2ノードクラスターでは、ポーリングにはシングルピアのみがあります。 *POLLING_TARGET* が定義されていない場合、どちらかのノードで障害が発生すると (たとえば、ノードがリブートされた、ノードのすべてのインターフェイスがダウンしているなど)、IP 監視が障害となり、動作可能ノードのすべてのサブネットがダウンしているとみなされます。そして、動作可能ノードで動作しているパッケージが障害となります。

そのため、シングルピアしか存在しない2ノードクラスターでは、ピアポーリングは適切ではありません。このような場合は、1つの LAN 障害が他の LAN のポーリングに影響しないように、必ずポーリングターゲットを定義する必要があります。

リンクレベルおよび IP レベルの障害の報告

リンクレベルまたは IP レベルでは、あらゆる障害が発生する可能性があります。障害がリンクレベルの監視と IP 監視のどちらで検出されるかによって、`cmviewcl (1m)` の出力でなされる報告が多少異なります。

以下に、障害がリンクレベルで検出された場合の `cmviewcl -v` の出力例を示します。

```
Network_Parameters:
INTERFACE      STATUS                PATH                NAME
PRIMARY        down (Link and IP)    0/3/1/0             eth2
PRIMARY        up                    0/5/1/0             eth3
```

`cmviewcl -v -f line` は、同じ障害を以下のような行形式で表示します。

```
node:gary|interface:lan2|status=down
node:gary|interface:lan2|disabled=false
node:gary|interface:lan2|failure_type=link+ip
```

以下に、障害が IP 監視によって検出された場合の `cmviewcl -v` の出力例を示します。

```
Network_Parameters:
INTERFACE      STATUS                PATH                NAME
PRIMARY        down (IP only)        0/3/1/0             eth2
PRIMARY        up                    0/5/1/0             eth3
```

`cmviewcl -v -f line` は、同じ障害を以下のような行形式で表示します。

```
node:gary|interface:lan2|status=down
node:gary|interface:lan2|disabled=false
node:gary|interface:lan2|failure_type=ip_only
```

パッケージ切り替えと再配置可能 IP アドレス

パッケージ切り替えでは、パッケージが新しいシステムに移動します。すべてのノードが同じサブネット上にある最も一般的な構成では、パッケージ IP(再配置可能 IP)。「[定常 IP アドレス](#)

および再配置可能 IP アドレスと監視対象サブネット」(54 ページ) を参照) も移動されます。また、新しいシステムではサブネットが正しく構成され動作していなければなりません。そうでなければ、パッケージは起動できません。

注記: ルーターで結合された複数のサブネットにまたがったクラスターを構成することができます。この構成では、一部のノードがあるサブネットを利用し、別のノードが別のサブネットを利用します。これを、クロスサブネット構成といいます。このような環境では、あるサブネット上のノードから別のサブネット上のノードにフェイルオーバーするようにパッケージを構成できるため、パッケージが開始するように構成されている各サブネットには、再配置可能アドレスを構成する必要があります。「クロスサブネットフェイルオーバーについて」(118 ページ)、特に「アプリケーションの配備への影響」(119 ページ) を参照してください。

パッケージ切り替えが行われると、TCP 接続が切断されます。TCP アプリケーションは、再接続して接続を回復する必要があります。これは自動的に処理されません。パッケージが複数のサブネット (パッケージ構成ファイル内の `monitored_subnet` で指定される) に依存している場合、通常はそのすべてのサブネットがターゲットノード上で使用可能になってから、パッケージを起動しなければなりません。(クロスサブネット構成では、ノード上に構成され、パッケージ構成ファイルで監視対象サブネットとして指定されているすべてのサブネットが使用可能でなければなりません。)

再配置可能 IP アドレスの切り替えを、[図 11](#)および[図 12](#)に示します。

同じサブネット上での切り替え後のアドレス解決メッセージ

ローカル切り替えまたはリモート切り替えにより、再配置可能 IP アドレスが新しいインターフェイスへ移動される際に、ARP メッセージがブロードキャストされ、IP アドレスとリンク層アドレス間の新しいマッピングを通知します。ARP メッセージは、移動された各 IP アドレスに対して送信されます。ブロードキャストされた ARP メッセージを受信するすべてのシステムは、関連する ARP キャッシュエントリをアップデートして、変更を反映する必要があります。現在、ARP メッセージは、IP アドレスが新しいシステムに追加されるときに送信されます。ARP メッセージは、ARP 要求の形で送信されます。ARP 要求メッセージの送信者と受信者のプロトコルアドレスフィールドは、ともに同じ再配置可能 IP アドレスに設定されます。これにより、メッセージを受信するノードが応答を送信しないようにすることができます。

IPv4 とは異なり、IPv6 のアドレスでは、近隣ノードのリンク層アドレスを調べるために、NDP メッセージを使います。

VLAN 構成

Serviceguard クラスターでは仮想 LAN(VLAN) 構成がサポートされます。

VLAN について

VLAN は、ネットワークノードを、物理的な場所とは無関係に、論理的にグループ化するための技術です。

VLAN は 1 つの物理 LAN を、複数の論理 LAN セグメントまたはブロードキャストドメインに分割するために使うことができます。これによりブロードキャストのトラフィックが減少するとともに、ネットワークの性能とセキュリティが向上し、管理が容易になります。

1 つの物理 LAN インターフェイスから、個別の IP アドレスを持つ複数の VLAN インターフェイスを構成できます。アプリケーションにとっては、これらの VLAN インターフェイスは、通常のネットワークインターフェイス (NIC) のように見えます。VLAN インターフェイスの構成に関しての詳細は、使用している Linux ディストリビューションのマニュアルを参照してください。

Linux VLAN のサポート

VLAN インターフェイスは、クラスター内のデータネットワークとしてだけでなく、ハートビートとしても使うことができます。ネットワークマネージャーはクラスター内に構成されている VLAN インターフェイスの状態を監視し、障害を検出した場合には、VLAN インターフェイス

のリモートフェイルオーバーを実行します。VLAN インターフェイスの障害は、通常、下層にある物理 NIC ポートまたはチャンネルボンディングインターフェイスの故障によって発生します。

構成上の制限事項

Linux では、1 つの物理 NIC ポートに対して最大 1024 個の VLAN が構成できます。このような構成に対応するためには、大量のシステムリソースが必要になります。クラスターの各ノードで多数のネットワークインターフェイスが構成されていると、Serviceguard の性能が低下します。このような問題を避けるために、Serviceguard には以下の制限があります。

- ノード当たり最大 30 個のネットワークインターフェイスがサポートされます。インターフェイスは、物理 NIC ポート、VLAN インターフェイス、チャンネルボンディング、あるいはこれらの組み合わせとすることができます。
- ポートベースの VLAN と IP サブネットベースの VLAN だけがサポートされます。Serviceguard は TCP/IP 以外のトランスポートプロトコルをサポートしていないため、プロトコルベースの VLAN はサポートされません。
- 各 VLAN インターフェイスには、固有のサブネット内の IP アドレスが割り当てられる必要があります。
- WAN クラスターでは、VLAN はサポートされません。

その他のハートビート要件

VLAN 技術では、ネットワークを非常に柔軟に構成することができます。このような環境で Serviceguard の信頼性と可用性を維持するために、クラスターで VLAN を使う場合には、ハートビートのルールが以下のように厳しくなっています。

1. 単一障害点を避けるために、VLAN のハートビートネットワークは、別の物理 NIC またはチャンネルボンディング上に構成する必要があります。
2. ハートビートは、VLAN を含め、すべてのクラスターネットワークでを使用することをお勧めします。
3. VLAN を使っており、ハートビートネットワークで VLAN を使わないことにした場合は、クラスター構成ファイルで指定したその他すべての物理ネットワークまたはチャンネルボンディングでハートビートを使用することをお勧めします。

データストレージ用のボリュームマネージャー

ボリュームマネージャーでは、個別のディスクパーティションよりも柔軟なディスクストレージボリュームを作成できます。このボリュームは、単一のシステムや、HA クラスターで使うことができます。HP Serviceguard for Linux では、冗長性のあるストレージグループの作成に Linux Logical Volume Manager (LVM) を使います。ここでは、LVM によるボリューム管理の概要を説明します。Serviceguard パッケージで使うボリュームグループ、論理ボリューム、ファイルシステムの構成についての詳細は、第 5 章の「[論理ボリュームインフラストラクチャの作成](#)」(132 ページ)を参照してください。

HP Serviceguard for Linux でサポートされている共有データストレージはディスクアレイで、ハードウェアで冗長性のあるストレージを構成します。

ディスクアレイでは、ストレージの基本要素は LUN です。LUN では、RAID1 または RAID5 によりストレージの冗長性がすでに確保されています。LUN を使う前に、fdisk を使ってパーティションを作成する必要があります。

LVM では、1 つ以上のボリュームグループ内のストレージを操作します。ボリュームグループは、個別の物理ボリュームをグループ化することによって構築されます。物理ボリュームは、ディスクパーティション、または後述のように物理ボリュームとしてマークされた LUN です。

pvcreate コマンドを使うと、LUN を物理ボリュームとして扱うことができます。その後、vgcreate コマンドを使って、1 つ以上の物理ボリュームからボリュームグループを作成します。ボリュームグループはいったん構成すると、異なるサイズやタイプの論理ボリュームに細分化することができます。クラスター内のアプリケーションが使うファイルシステムやデータベースは、これらの論理ボリュームの上に作成されます。Serviceguard クラスターでは、アプ

リケーションの起動時に、パッケージ制御スクリプトによってボリュームグループがアクティブ化されます。そして、アプリケーションの停止時に、パッケージ制御スクリプトによってボリュームグループが非アクティブ化されます。

アレイ上のストレージ

図 26 に、ストレージアレイに構成された LUN を示します。物理ディスクは、アレイユーティリティプログラムにより、オペレーティングシステムから参照可能な論理ユニット (LUN) として構成されています。

図 26 LUN として組み合わされている物理ディスク



注記: LUN の定義は通常、ディスクアレイ製造メーカーが提供するユーティリティプログラムを使って行います。アレイは機種によって大きく異なるため、ご使用のストレージユニットに付属のドキュメントを参照してください。

マルチパス構成については、「[ストレージのマルチパス](#)」(75 ページ) を参照してください。

ディスクの監視

各パッケージ構成には、パッケージの起動時にアクティブ化されるディスクに関する情報が含まれます。監視機能を使っている場合、パッケージの起動時にディスクの稼働状態がチェックされます。ディスクが使用不可能な場合はパッケージの起動に失敗します。

この場合、パッケージは別のノードで再起動されることになります。`auto_run` を `yes` に設定した場合、起動の条件をすべて満たしていれば、別の使用可能なノードでパッケージが起動されます。`auto_run` を `no` に設定した場合は、パッケージを別のノードで起動することなく、そのパッケージが停止されます。

ディスクモニターの構成プロセスについては、「[ディスク監視構成の作成](#)」(184 ページ) で説明します。

LVM についての詳細

Serviceguard パッケージで使うボリュームグループ、論理ボリューム、ファイルシステムの構成についての詳細は、第 5 章の「[論理ボリュームインフラストラクチャの作成](#)」の項を参照してください。

Linux LVM についての基本的な説明は、Linux Documentation Project ページ (<http://www.lldp.org>) に掲載されている『Logical Volume Manager HOWTO』を参照してください。

Persistent Reservation について

Serviceguard for Linux パッケージは、可能な場合には常に Persistent Reservation (PR) を使用して LUN へのアクセスを制御します。PR は SCSI Primary Commands バージョン 3 (SPC-3) 規格

で定義されたもので、I/O イニシエーターを登録する手段を提供し、LUN デバイスにアクセス可能なユーザー (すべて、登録者すべて、登録者のうち 1 人のみ) およびアクセス可能な方法 (読み取り専用、書き込み専用) を指定します。

ボリュームグループの排他的なアクティブ化では、下層にある LUN への不正アクセスを防止できませんが、PR はこれとは異なり、LUN レベルでアクセスを制御します。登録情報および予約情報はデバイスに保存され、ファームウェアによって施行されます。この情報は、デバイスのリセットおよびシステムのリブート後も保持されます。

注記: PR は、ボリュームグループのアクティブ化保護とは共存しており、依存はしていません。ボリュームグループのアクティブ化保護を有効にする手順に従い、アクティブ化保護を引き続き設定する必要があります。以下の規則と制限事項に従い、PR はクラスターの LUN に適用されます。ボリュームグループに LUN が構成されているかどうかは関係ありません。

PR の利点は次のとおりです。

- 一貫した動作
排他アクティブ化を行う方法はボリュームマネージャーごとに異なる可能性があります (まったくこの処理を行わない場合もある)、PR はデバイスレベルで行われ、排他アクティブ化のためにボリュームマネージャーに依存することはありません。
- パッケージは、ボリュームマネージャーとは無関係に LUN デバイスへのアクセスを制御できます。
Serviceguard では ASM マネージャーがサポートされているため、アプリケーションでこれらのプロトコルを使っているパッケージは、ボリュームマネージャーを使うことなくストレージデバイスに直接アクセスできます。

規則と制限事項

Serviceguard は、以下の制限に基づき、LUN ストレージを使うパッケージに対して自動的に PR を実施します。

- LUN デバイスは、PR をサポートし、かつ SPC-3 仕様に沿ったものでなければなりません。
- PR は、従来のマルチノードパッケージでは利用できません。
PR は、モジュール方式のマルチノードパッケージと、モジュール方式および従来方式のフェイルオーバーパッケージで利用できます。
 - モジュール方式のマルチノードパッケージのインスタンスはすべて、PR を使用できる状態でなければなりません。この条件を満たさない場合、すべてのインスタンスに対して PR が無効になります。
- パッケージは、仮想化されたデバイスだけでなく、実際のデバイスにもアクセスできなければなりません。
- VMware ゲストであるノードが含まれるクラスターは、以下の制限の下、PR を使用できません。
 - 同じクラスター内でノードとして稼働している 2 つ以上の VMware ゲストは、同じホスト上で稼働できません。
(個別のホスト上に存在する VMware ゲストであれば、複数の VMware ゲストを 1 つのクラスターに構成できます。また、異なるクラスター内に存在するゲストであれば、複数のゲストを 1 つのホストに構成できます。)
 - VMware ゲスト上で稼働しているパッケージは、ベース物理 LUN にアクセスする手段として Raw デバイスマッピングを使う必要があります。

Serviceguard と VMware 仮想マシンの併用についての詳細は、<http://www.hp.com/go/linux-serviceguard-docs> にあるホワイトペーパー『Using Serviceguard for Linux with VMware Virtual Machines』を参照してください。

- △ **注意:** Serviceguard は、パッケージの通常の起動と停止の際、またはパッケージのフェイルオーバーの際に、登録と予約の実施と取り消しを行います。Serviceguard には、破壊的なクラスタ障害が発生した場合に登録を消去するスクリプトも用意されています。そのような障害時にはこのスクリプトを必ず実行してください。これを怠ると、LUN デバイスが使用不能になる可能性があります。詳細は、「[破局的な障害後の Persistent Reservation の取り消し](#)」(237 ページ)を参照してください。

Persistent Reservation の動作

Persistent Reservation (PR) を有効化またはアクティブ化するために設定を行う必要はありません。実際には、ユーザー自身でクラスタレベルまたはパッケージレベルで PR を有効または無効にすることはできません。上記の[規則と制限事項](#)に基づいて Serviceguard がクラスタとパッケージごとに決定を下します。

ユーザーが `cmapplyconf (1m)` を実行して新しいクラスタを構成する場合や、新しいノードを追加する場合、Serviceguard によって変数 `cluster_pr_mode` が `pr_enabled` または `pr_disabled` に設定されます。

- `ENABLED` は、パッケージは原則 PR を使用できるが、実際には「[規則と制限事項](#)」に挙げられている条件を満たす場合にのみ使用可能であることを意味します。
- `DISABLED` は、パッケージで PR を使用できないことを意味します。

`cluster_pr_mode` の設定は、`cmviewcl -f line` の出力で確認できます。次に例を示します。

```
...
cluster_pr_mode: pr_enabled
```

注記: ユーザーが `cluster_pr_mode` の設定を変更することはできません。

PR を使用する権限がパッケージに与えられている場合、Serviceguard が自動的に `sg_persist` コマンドを使ってパッケージの起動時にパッケージの LUN 用に登録と予約の実施と取り消しを行い、パッケージの停止時にそれらの取り消しを行います。このコマンドは、Red Hat 5、Red Hat 6、SUSE 11 のいずれでも使用できます。また、マンページもあります。

Serviceguard は、パッケージの LUN デバイスに対してタイプ WERO (Write Exclusive Registrants Only) の PR を実施します。これにより、登録されているかどうかにかかわらず、すべてのイニシエーターに読み取り権限が与えられ、書き込み権限は登録されているイニシエーターだけに与えられます (WERO は SPC-3 規格で定められています)。

パッケージを実行する、各ノード上のイニシエーターはすべて、`node_pr_key` と呼ばれる同一の `PR` キーを使って LUN デバイスに登録します。クラスタ内の各ノードには、一意の `node_pr_key` が与えられます。このキーは、`cmviewcl -f line` の出力で確認できます。以下に例を示します。

```
...
node:bla2|node_pr_key=10001
```

フェイルオーバーパッケージの起動時に、まずベース LUN デバイスから既存の PR キーと予約が削除され、その後、パッケージが起動されるノードの `node_pr_key` が各 LUN に登録されます。

マルチノードパッケージの場合、パッケージの最初のインスタンスによってベース LUN に PR 予約が行われ、新しいノードでパッケージが起動されるたびに適切な `node_pr_key` が登録されます。ノードに障害が発生すると、他のノード上で稼働しているパッケージインスタンスが、障害が発生したノードの登録を削除します。

特定のパッケージで PR が有効になっているかどうかは、`cmgetpkgenv (1m)` で確認できます。以下に例を示します。

```
cmgetpkgenv pkg1
```

```
...
```

```
PKG_PR_MODE="pr_enabled"
```

障害への応答

Serviceguard は、さまざまな障害に対して個別の方法で応答します。大部分のハードウェア障害については、応答をユーザーが構成することはできませんが、パッケージ障害とサービス障害については、ユーザーは一定の範囲内でシステムの応答を選択できます。

ノード障害発生時のリブート

Serviceguard クラスター内で発生した障害に対する最も顕著な応答は、システムリブートです。これにより、パッケージはすぐに別のノードに移動し、データの完全性が保護されます。

リブートは、クラスターノードが、事前に定義された一定の時間クラスターメンバーの過半数と通信できなくなると発生します。また、カーネルのハングや、クラスターデーモン (`cmclld`) の障害などの状況でも発生します。この現象が発生すると、次のメッセージがコンソール上に表示されることがあります。

```
DEADMAN: Time expired, initiating system restart.
```

このケースについては、「[ノードがタイムアウトしたときの動作](#)」でさらに詳しく説明しています。「[クラスターデーモン: cmclld](#)」(30 ページ) も参照してください。

リブートは、特殊な状況で、Serviceguard 自身によっても引き起こされます。「[パッケージ障害とサービス障害への応答](#)」(70 ページ) を参照してください。

ノードがタイムアウトしたときの動作

各ノードは、他のすべてのノードにハートビートメッセージを送信します。送信間隔は、構成された `MEMBER_TIMEOUT` の値の 4 分の 1 または 1 秒のいずれか小さいほうです。

`MEMBER_TIMEOUT` は、クラスター構成ファイルで設定します。「[クラスター構成のパラメーター](#)」(82 ページ) を参照してください。ハートビート間隔は直接設定することはできません。ノードが `MEMBER_TIMEOUT` で設定された時間内にハートビートメッセージを送信できなかった場合、ハートビートメッセージを送信していないノードを除いてクラスターが再編成されます。

ノードが別のノードの障害を検出した場合 (つまり、`MEMBER_TIMEOUT` マイクロ秒以内にハートビートメッセージが到着しない場合)、以下のイベントシーケンスが発生します。

1. ノードはその他のノードに連絡を取り、障害が発生したノードを除いてクラスターを再構成しようとします。
2. 残りのノードが過半数である場合、または残りのノードがクラスターロックを取得できる場合、これらのノードは障害が発生したノードを除いて新しいクラスターを形成します。
3. 残りのノードが過半数でない場合、または残りのノードがクラスターロックを取得できない場合、これらのノードは停止します (システムリセット)。

例

状況。 2 台のノードからなるクラスターを考えます。Package1 が SystemA で動作しており、Package2 が SystemB で動作しています。ボリュームグループ `vg01` が、SystemA 上で排他的にアクティブ化されています。ボリュームグループ `vg02` は、SystemB で排他的にアクティブ化されています。パッケージの IP アドレスは、SystemA と SystemB にそれぞれ割り当てられています。

障害。 ハートビート用とデータトラフィック用に、1 つの LAN だけが構成されていました。運用中に大量のアプリケーショントラフィックでネットワークの帯域幅が独占され、ハートビートパケットが送受信できなくなりました。

SystemA は SystemB からのハートビートメッセージを受信できないため、SystemA は、1 ノードのクラスターとして再編成しようとします。同様に、SystemB は SystemA からのハートビートメッセージを受信できないため、SystemB も 1 ノードのクラスターとして再編成しようとします。選出プロトコルで各ノードは自分自身に投票し、それぞれのノードに投票の 50 パーセントが割り当てられます。両方のノードが投票の 50 パーセントを得ているため、両方のノードがクラスターロックを獲得しようとします。ロックを取得できるのは 1 台のノードだけです。

結果。 SystemA がクラスターロックを取得したとします。SystemA は 1 ノードクラスターとして再編成します。再編成の後、SystemA は、既存のクラスターノードで動作するように構成されていたすべてのアプリケーションが動作していることを確認します。SystemA が、クラスターで Package2 が動作していないことを検出すると、Package2 が SystemA で動作するように構成されていれば、Package2 を起動します。

SystemB は、クラスターロックの取得に失敗したことを認識するため、クラスターを再編成することができません。Package2 に関連するすべてのリソース (ボリュームグループ vg02 への排他的アクセスや、Package2 の IP アドレス) をできるだけ早く解放するために、SystemB は停止 (システムリセット) します。

注記: `/etc/rc.config.d/cmcluster ($SGAUTOSTART)` 内の `AUTOSTART_CMCLD` にゼロを設定すると、ノードが復帰しても、そのノードはクラスターへの参加を試みません。

クラスターのフェイルオーバーについての詳細は、<http://www.hp.com/go/linux-serviceguard-docs> -> [White papers] にあるホワイトペーパー『Optimizing Failover Time in a Serviceguard Environment (version A.11.19 or later)』を参照してください。トラブルシューティング情報については、「[MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスターの再編成](#)」(244 ページ) を参照してください。

ハードウェア障害への応答

システムのパニックまたは SPU 回路の物理的破壊など、システムに重大な問題が発生した場合、Serviceguard はノード障害を認識し、そのノードで現在動作しているパッケージをクラスター内の別の引き継ぎノードへ移行します。(システムマルチノードパッケージとマルチノードパッケージはフェイルオーバーしません。)

各パッケージの移行後の位置は、そのパッケージの構成ファイルによって決定されます。この構成ファイルには、パッケージの一次ノードとその代替ノードが記述されています。パッケージを別のノードへ移行しても、プログラムカウンターは引き継がれません。移行したパッケージのプロセスは最初から再起動します。障害発生後にアプリケーションを迅速に再起動させるには、そのアプリケーションがクラッシュの影響を受けにくい性質 (crash-tolerant) を備えていなければなりません。つまり、パッケージのすべてのプロセスは、このような場合の再起動を検出できるように作成されていることが必要です。これは、通常のシステムクラッシュ後の再起動に要求されるアプリケーション設計と同じです。

LAN インターフェイス障害が発生した場合は、ボンディングにより IP メッセージのバックアップパスが用意されます。ハートビート LAN インターフェイスに障害が発生した場合、冗長ハートビートが構成されていなければ、ノードは障害となってリブートします。監視対象データ LAN インターフェイスで障害が発生した場合は、パッケージで `node_fail_fast_enabled` (詳細は、「[パッケージの構成: 次の手順](#)」(120 ページ) を参照) に `yes` が設定されている場合に限り、ノードが障害となってリブートします。それ以外の場合には、その LAN インターフェイスを使っていたパッケージは停止され、可能なら別のノードに移行されます (LAN がすぐに回復しない限り。「[サービスやサブネットの異常終了時、または汎用リソースや依存関係が満たされない場合](#)」(51 ページ) を参照してください)。

ディスク監視を使うと、さらに保護を行うことができます。パッケージがディスクの状態に依存するように構成すると、ディスク監視プログラムによって障害が報告されたときに、パッケージを他のノードにフェイルオーバーさせることができます。「[ディスク監視構成の作成](#)」(184 ページ) を参照してください。

Serviceguard は電源異常には直接応答しません。ただし Serviceguard では、各クラスター構成要素の電源障害はその構成要素自体の障害とみなされるため、結果的には適切な切り替え動作が行われます。電源保護は、HP がサポートする無停電電源装置 (UPS) によって行います。

パッケージ障害とサービス障害への応答

デフォルトの場合、パッケージ、またはパッケージの汎用リソースやサービスの障害、あるいはパッケージ内のサービスの障害が発生すると、`stop` パラメーターを使って制御スクリプトが実行されることにより、パッケージはシャットダウンされ、その後代替ノード上で再起動されます。パッケージに他のパッケージへの依存関係が定義されていて、そのパッケージで障害が発生した場合には、依存している側のパッケージも障害状態になります。

このデフォルト動作は、必要に応じて変更できます。つまり、パッケージの移行が実行される前に、ノードが停止(システムリセット)するように指定できます。それには、パッケージ構成ファイル内の `failfast` パラメーターを設定します。

パッケージのシャットダウンでハングして、ノードの状態が不明になる可能性がある場合は、この `failfast` オプションにより迅速なフェイルオーバーが可能となります。また、ノードはリブートによってクリーンアップされます。ただし、システムリセットでは、ノードの**すべての**パッケージが突然停止されることに注意してください。

パッケージ構成ファイル内の `failfast` パラメーターの設定によって、パッケージまたはリソースが障害になったときのパッケージとノードの動作が決まります。

- パッケージ構成ファイル内で `service_fail_fast_enabled` (172 ページ) を `yes` に設定すると、そのサービスに障害があった場合に、Serviceguard はノードをリブートします。
- パッケージ構成ファイル内で `node_fail_fast_enabled` (164 ページ) を `yes` に設定すると、パッケージに障害があった場合に、Serviceguard はパッケージが動作しているノードを停止 (リブート) します。

詳細は、「[パッケージ構成のプランニング](#)」(92 ページ)と第6章(157 ページ)を参照してください。

パッケージ障害と汎用リソース障害への応答

汎用リソースが構成され実行中のパッケージでは、リソースで障害が発生すると、パッケージの型に合わせて適切な対応を行うよう Serviceguard パッケージマネージャーに指示が出されます。

フェイルオーバーパッケージの場合、リソースの障害が発生したノード上のパッケージは停止し、使用可能な代替ノード上で起動します。マルチノードパッケージの場合、汎用リソースに障害が発生すると、障害が発生したノード上のパッケージは停止するだけです。

- シンプルリソースの場合、リソースの障害は、監視スクリプトが `cmsetresource` コマンドを使用してリソースのステータスを「`down`」に設定するトリガーとなります。
- 拡張リソースの場合は、監視スクリプトで取得した値を `cmsetresource` コマンドで設定できます。

Serviceguard パッケージマネージャーは、リソースが構成されているパッケージでそのリソースについて設定されている `generic_resource_up_criteria` に対してこの値を評価します。設定されている値 (`current_value`) が `generic_resource_up_criteria` の条件を満たさない場合、そのノードの汎用リソースには「`down`」のマークが付きます。

注記: あるノードでシンプルリソースが停止すると、そのノード上ではそのリソースを使用しているすべてのパッケージでリソースが停止します。一方、拡張リソースの場合は、`generic_resource_up_criteria` に依存しているため、リソースが、1つのノード上のあるパッケージではアップになり、別のパッケージではダウンになることがあります。

また、汎用リソースが構成される稼働中のパッケージでは、次の状態が発生します。

- パッケージで `evaluation_type` が「`before_package_start`」に構成されている汎用リソースに障害が発生しても、そのパッケージのノード切り替えは無効になりません。
- パッケージで `evaluation_type` が「`during_package_start`」に構成されている汎用リソースに障害が発生すると、そのパッケージのノード切り替えは無効になります。

「切り替え動作とフェイルオーバー動作の選択」(96 ページ) で、適切なフェイルオーバー動作を選択する上での推奨事項を説明しています。

「汎用リソースを構成するパラメーター」(97 ページ) を参照してください。

サービスの再起動

障害発生後に、サービスをローカルシステムで再起動できます。これを行うには、パッケージ制御スクリプトで各サービスの再起動回数を指定します。サービスの起動時に、このサービスの環境で変数 `service_restart` が設定されます。サービスは、実行中にこの変数を調べて、障害発生後に再起動されたかどうかをチェックします。再起動されている場合は、クリーンアップなどの適切な処理を行います。

ネットワーク通信障害

クラスターの重要な要素として、ネットワークの状態が挙げられます。各ノードはクラスターを継続的に監視しながら、他のノードから送られてくるハートビートメッセージをチェックして、すべてのノードが互いに通信できることを確認します。設定時間内にノードがこのようなメッセージを受信しなかった場合、ノードのタイムアウトが発生し、クラスターの再編成が行われます。その後ハートビートメッセージを受信しなかった場合はリブートが実行されます。「ノードがタイムアウトしたときの動作」(68 ページ) を参照してください。

4 HA クラスターのプランニングと文書化

Serviceguard クラスターの構築は、その構成に関するすべてのハードウェアおよびソフトウェアのコンポーネントに関する情報を集めて記録する、計画の段階から始めます。

この章は、以下のプランニング作業に役立ちます。

- プランニング全般
- ハードウェアのプランニング (74 ページ)
- 電源のプランニング (76 ページ)
- クラスターロックのプランニング (77 ページ)
- ボリュームマネージャーのプランニング (78 ページ)
- クラスター構成のプランニング (79 ページ)
- パッケージ構成のプランニング (92 ページ)

付録C(264 ページ)に、未記入のワークシート一式があります。このワークシートは、構成の要点を文書として記録しておくのに便利です。

注: プランニングとインストールの作業は重複する部分が多いので、実際の構成を開始する前に、ワークシートが完成していなくても構いません。この場合は、構成作業を行いながら、未記入の項目を記入してください。

構成や保守作業については、以降の章で詳しく説明します。

プランニング全般

ハイアベイラビリティ(高可用性)の目的を十分理解することにより、ハードウェアの必要条件の定義とシステムの設計を迅速に行うことができます。全般的なプランニングの手引きとして以下の質問を使用してください。

1. 障害が発生した場合でも、引き続き使用できないなければならないアプリケーションはどれか。
2. 上記のアプリケーションを使用するために必要なシステムリソース(処理能力、ネットワーク、SPU、メモリ、ディスクスペース)は何か。
3. 通常の運用時にクラスター内のノード間に上記のリソースをどのように分配するか。
4. 発生する可能性のある障害のすべての組み合わせ、特にノード障害が発生した場合に、上記のリソースをクラスター内のノード間にどのように分配するか。
5. クラスターの定期保守作業時に、リソースをどのように分配するか。
6. ネットワークの必要条件は何か。すべてのネットワークとサブネットは使用可能か。
7. すべての単一障害点を取り除いたか。例:
 - ネットワーク障害
 - ディスク障害
 - 電気系統の障害
 - アプリケーション障害

Serviceguard のメモリ要件

Serviceguard には、ロック可能メモリがおよそ 15.5MB 必要です。

拡張のプランニング

初めてクラスターを設定する際、初期構成用としてのノードセットを設定してパッケージのグループを定義します。後でノードやパッケージを追加したり、共有データストレージ用として

別のディスクハードウェアを使用できます。クラスターを**停止しないで**拡張するには、初期構成のプランニングを慎重に行うことが必要です。以下のガイドラインに従ってください。

- Maximum Configured Packages パラメーター (この章の「[クラスター構成のプランニング](#)」(79 ページ) で後述します) は、今後追加する予定のパッケージの数を十分に上回る値に設定してください。
- クラスターを稼働させたまま追加する予定のパッケージでネットワークが必要になる場合は、クラスター構成に事前にそのネットワークを構成しておいてください。「[LAN 情報](#)」(74 ページ) を参照してください。

クラスターの稼働中にクラスター構成を動的に変更する方法については、[第7章「クラスターとパッケージの管理」](#) (186 ページ) を参照してください。

Serviceguard と VMware 仮想マシンの併用

HP Serviceguard for Linux は、VMware ESX サーバー上に作成された Linux 仮想マシンへの配備が保証されています。ここでは、アプリケーションの高可用性を維持するために、ESX サーバー上で稼働する VMware 仮想マシンおよび物理マシンを使用する、Serviceguard for Linux クラスターのさまざまな構成を説明します。

注記: Serviceguard は ESX ホストについてではなく VMware ESX ゲストについて動作が保証されており、仮想マシン自体ではなくアプリケーションの高可用性を提供します。

VMware は VMware HA と呼ばれる高可用性 (HA) クラスター化製品を提供しており、これは、障害からの保護を一定の程度で実現します。VMware HA は、物理サーバーの障害のみを検出する単純なモデルを使用します。このような障害が検出されると、仮想マシンは障害の発生したサーバーから移動し、VMware を実行している別のサーバー上で再起動されます。仮想マシンで Serviceguard for Linux を実行すると、この保護のレベルを大きく向上させることができます。Serviceguard は、以下のようなさまざまな障害の発生時に、アプリケーションをフェイルオーバーします。

- アプリケーションの障害
- アプリケーションが必要とするネットワークの障害
- ストレージの障害
- オペレーティングシステムの「ハング」または仮想マシン自体の障害
- 物理マシンの障害

さらに、次のような利点もあります。

- VM ゲストの計画的ダウンタイムおよび予期しないダウンタイムを最小限に抑えます。
- Serviceguard for Linux のフェイルオーバーは、VMware HA よりも高速です。
- Serviceguard for Linux の段階的アップグレード機能により、計画的ダウンタイムをさらに短縮できます。

Serviceguard は各ノード上で稼働しながら、パッケージにまとめられたアプリケーションを管理するように設計されているため、VM を Serviceguard ノードとして構成することで Serviceguard と VMware VM を統合できます。このような構成では、仮想マシンは Serviceguard クラスターのメンバーとなり、クラスター内の他の物理的ノードまたは VM ノード間でアプリケーションパッケージのフェイルオーバーが可能になります (Serviceguard は VM ゲスト内で実行されます)。

クラスター構成の選択肢

VM ノードを含む Serviceguard クラスターは、以下のような構成にすることができます。

- 同一ホスト上の仮想マシン (1 つのボックス内でのクラスター、推奨されません)
- 別々のホスト上の仮想マシン

- 仮想マシンと物理的ノード
- 上記すべての組み合わせ

詳細は、<http://www.hp.com/go/linux-serviceguard-docs> にあるホワイトペーパー『Using Serviceguard for Linux with VMware Virtual Machines』を参照してください。

ハードウェアのプランニング

ハードウェアのプランニングにおいては、物理的なハードウェア自身を調べなくてはなりません。1つの有益な手順は、アダプターカードとバス、ケーブル接続、ディスクおよび周辺装置を表すハードウェア構成の概略図を書くことです。

ハードウェア用ワークシート(264 ページ)に情報を記録するのも有益です。どのデバイスアダプターがどのスロットを占有するかを指定し、クラスター構成を作成しながら詳細を更新します。ノード (サーバー) ごとに 1つのフォームを使ってください。

SPU 情報

SPU 情報には、クラスターで使っているサーバーシステムの基本特性が含まれています。ハードウェア用ワークシート(264 ページ)に以下の内容を記録することをお勧めします。

Server Series Number

シリーズ番号を記入します (DL980 G7 など)。

Host Name

システムでホスト名として使用される名前を記入します。

Memory Capacity

メモリの容量を MB 単位で記入します。

Number of I/O slots

スロットの数を記入します。

LAN 情報

各サブネットにつき最低 1つの LAN インターフェイスが必要ですが、ネットワークの単一障害点を無くすには、最低 2つの LAN インターフェイスが必要です。

クライアントデータ用に使用するサブネットを含むすべてのサブネットにハートビートを構成することをお勧めします。

LAN インターフェイスごとに以下の情報を収集してください。

Subnet Name

サブネットの IP アドレスです。ハートビートの IP アドレスは、各ノードの同じサブネット上になければなりません。

Interface Name

このノードがサブネットへのアクセスに使用する LAN カードの名前。カードのインストール後は、ifconfig コマンドを使うとこの名前が表示されます。

IP Address

このインターフェイスで使用する IP アドレスです。

IPv4 アドレスは、次の形式の、4つの数をピリオドで区切った文字列です。

nnn.nnn.nnn.nnn

IPv6 アドレスは、次の形式の、8個の 16進数をコロンで区切った文字列です。

xxx:xxx:xxx:xxx:xxx:xxx:xxx:xxx

IPv6 のアドレス形式についての詳細は、付録 D (268 ページ) を参照してください。

Kind of LAN Traffic

サブネットの目的です。指定できる項目を以下に示します。

- Heartbeat (ハートビート)
- Client Traffic (クライアントトラフィック)

リストに名前を付けて、ブリッジネットに所属するサブネットを明示してください。

この情報は、サブネットグループの作成と、クラスター構成ファイルおよびパッケージ構成ファイルで使われる IP アドレスの確認に使われます。

共有ストレージ

SCSI は最大 4 ノードのクラスターで利用でき、FibreChannel は最大 16 ノードのクラスターで利用できます。

FibreChannel

FibreChannel カードを使うと、ストレージが収められたディスクアレイに対して最大 16 ノードまで接続することができます。カードを装着し適切なドライバーをインストールすると、ストレージ上で構成された LUN はオペレーティングシステムからはデバイスファイルとして見えます。このデバイスファイルを使って、LVM ボリュームグループを構築できます。

注記: FibreChannel HBA デバイスドライバーと Linux Device Mapper では、マルチパス機能がサポートされています。詳細は、ストレージデバイスのマニュアルを参照してください。

「[ストレージのマルチパス](#)」も参照してください。

ワークシートに、FibreChannel に接続されているストレージユニットの各 LUN に対応するデバイスファイルの名前を記入できます。

ストレージのマルチパス

マルチパスソリューションを実現するための方法は、クラスターに接続されているストレージサブシステムとサーバーのホストバスアダプター (HBA) に依存します。ストレージサブシステムと HBA に付属しているマニュアルを参照してください。

Fibre Channel に接続されたストレージでは、当社が提供する HBA ドライバーでマルチパス機能がサポートされている場合、それを使う必要があります。QLogic ドライバーについては、『HP StorageWorks Using the QLogic HBA driver for single-path or multipath failover mode on Linux systems application note』を参照してください。このドキュメントは、<http://www.hp.com> の検索ボックスに「qlogic multipath application」と入力することで見つかります。

注記: Linux の急速な進化により、マルチパス機構が変化したり、新しい機構が追加される可能性があります。Serviceguard for Linux は DeviceMapper マルチパス (DM-MPIO) を制限付きでサポートしています。最新情報については、このマニュアルの「はじめに」に記載しているアドレスの『HP Serviceguard for Linux Certification Matrix』を参照してください。

注記: md はソフトウェア RAID もサポートしていますが、この構成は現在の Serviceguard for Linux ではサポートされていません。

ディスク入出力情報

付録 C のハードウェア用のワークシートを使うと、ノード上の各ディスクデバイスアダプターに接続されている各ディスクについて以下の情報を記録できます。

Bus Type

バスの種類を記入します。使用できるバスは、SAS (Serial Attached SCSI) と FibreChannel です。

LUN Number

ストレージユニットで定義されている LUN の番号を記入します。

Slot Number

コンピューターのバックプレーンの、SCSI または FibreChannel インターフェイスカードを挿入するスロットの番号を記入します。

Address

バスのハードウェアバスの番号を記入します。この番号は、ホストパラメーターの数値部分であり、次のコマンドを使ってシステム上で表示できます。

```
cat /proc/scsi/scsi
```

Disk Device File

各 SCSI ディスクまたは LUN のディスクデバイスファイル名を記入します。

この情報は、LVM を使用してミラー化ディスクの構成を作成する際に必要です。また、ディスク構成についてできるだけ多くの情報を収集することも役立ちます。

以下のコマンドを使用することにより、使用可能なディスクに関する情報を入手できます。システムで他のユーティリティが提供されていることもあります。

- `ls /dev/sd*` (Smart Array クラスタストレージ)
- `ls /dev/hd*` (SCSI/FibreChannel 以外のディスク)
- `ls /dev/sd*` (SCSI ディスクと FibreChannel ディスク)
- `du`
- `df`
- `mount`
- `vgdisplay -v`
- `lvdisplay -v`

詳しい使用方法については、これらのコマンドのマニュアルを参照してください。ハードウェアをインストールして、システムをリブートした後、**すべてのノード**からコマンドを実行してください。上記の情報は、LVM やクラスタの構成を行う際に役立ちます。

ハードウェア構成用ワークシート

特定のクラスタのハードウェア構成を作成して記録する際に、ハードウェア構成ワークシート(264 ページ)を使うと便利です。このワークシートを必要な数だけコピーしてください。

電源のプランニング

設計時に考慮が必要なクラスタの電源が 2 つあります。1 つは電力線の電源で、もう 1 つは無停電電源装置 (UPS) の電源です。電源系統の障害が原因でクラスタが停止しないようにしなければなりません。

サーバー、大容量ストレージデバイス等のハードウェアには、独立した電源が 2 つまたは 3 つ搭載されていることがよくあり、1 つの電源や電源系統への給電が停止しても動作し続けられるようになっています。デバイスに冗長電源が搭載されている場合は、それぞれの電源を別の電源系統に接続してください。このようにすると、単一の電源系統に障害が発生してもクラスタ内の重要な装置が完全に停止することはありません。たとえば、クラスタ内の各装置に 3 つずつ電源が搭載されている場合、電源がクラスタの単一障害点とならないようにするためには、最低でも 3 つの電源系統が必要になります。電源が 1 つしかないハードウェアの場合は、半数以上のノードで同じ電源を使わないでください。また、ちょうど半数のノードの電源を 1 つの電源系統から取る場合は、この電源をクラスタロック LUN やクォーラムサーバーの電源に使用しないでください。使用すると、障害発生後にクラスタを再編成できなくなります。詳細は、「[クラスタロックのプランニング](#)」(77 ページ)を参照してください。

電源異常時に高可用性を実現するには、少なくとも各ノードの SPU とクラスタロックディスク (使用する場合) に対して、個別の UPS を使用してください。クォーラムサーバーまたは

クォーラムサーバークラスターを使用している場合、各クォーラムサーバーノードの電源はサービス対象の各クラスターの電源とは分離してください。ソフトウェアミラー機能を使用する場合は、異なる物理ボリュームグループ間で電源を共有しないでください。これにより、異なる電源に接続された、異なる I/O バス上にある物理ディスク間でミラー化を設定できます。混乱を防ぐため、ハードウェアユニットと電源ユニットに別々のユニット番号を付けてください。電源用ワークシートに、使用しているハードウェアユニットとそれらが接続されている電源を明記します。ワークシートに以下の項目を記入します。

Host Name

各 SPU のホスト名を記入します。

Disk Unit

各ディスクのディスクドライブのユニット番号を記入します。

Tape Unit

各バックアップ装置のテープのユニット番号を記入します。

Other Unit

その他のユニットの番号を記入します。

Power Supply

ホストまたは他の装置が接続されている UPS の電源のユニット番号を記入します。

SPU の最大電源容量だけでなく、UPS、電源回路、キャビネットの最大電源容量も守ってください。

電源構成用ワークシート

電源プランニングワークシート(264 ページ) は、特定の電源構成を作成して記録する際に便利です。このワークシートを必要な数だけコピーしてください。

クラスターロックのプランニング

クラスターロックの目的は、それまでクラスターを構成していたノードのちょうど半分で新しいクラスターを形成しようとした場合に、新しいクラスターが必ず 1 つだけ形成されるようにすることです。新しいクラスターが 1 つだけ形成され、そのクラスターのパッケージで指定されているディスクにこのクラスターだけがアクセスを許可されることが非常に重要です。クラスターロックとして、ロック LUN またはクォーラムサーバーを指定できます。クラスターロックについての詳細は、「[クラスターロック](#)」(35 ページ) を参照してください。

注記: 同一のクラスター内で、複数のタイプのロックを使うことはできません。

クラスターロックの要件

1 ノードのクラスターにはロックは不要です。2 ノードのクラスターではクラスターロックを使う必要があり、2 ノードより大きなクラスターでもロックを使用することをお勧めします。ノードが 4 つを超えるクラスターでは、クラスターロックとして使用できるのはクォーラムサーバーだけです。

ロック LUN および Quorum Server の構成についての詳細は、「[ロック LUN の設定](#)」(130 ページ)、「[ロック LUN の指定](#)」(143 ページ)、および <http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にある HP Serviceguard Quorum Server バージョン A.04.00 以降のリリースノートを参照してください。

拡張のプランニング

ノードが 5 つ以上あるクラスターでは、ロック LUN を使用できないことに注意してください。したがって、合計で 5 つ以上になるようにノードを追加する予定がある場合は、クォーラムサーバーを使う必要があります。

クォーラムサーバーの使用

Quorum Server については、「[クラスターロックとしてのクォーラムサーバーの使用](#)」(36 ページ)を参照してください。「[クラスターロック](#)」(35 ページ)も参照してください。

クォーラムサーバーには、次の特長があります。

- 最大 150 クラスター、合計 300 ノードから使用できる。
- サポートされる任意の数のノードで構成されるクラスターをサポートできる。
- サポートされる任意の数のノードで構成されるクラスターをサポートできる。
- 2 つまでのサブネット (一次と代替) でクラスターと通信できる。

- ① **重要:** クォーラムサーバーを使う予定の場合は、先に進む前に『HP Serviceguard Quorum Server Version A.04.00 Release Notes』を参照してください。これは、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] から入手できます。また、同じ場所にある Quorum Server のホワイトペーパーも参照してください。

クォーラムサーバー用ワークシート

クォーラムサーバー用ワークシート([265 ページ](#))を使うと、1 つまたは複数のクラスターで使うクォーラムサーバーを特定できます。以下の内容を記録することをお勧めします。

Quorum Server Host

クォーラムサーバーのホスト名です。

IP Address

クォーラムサーバーがクラスターノードと通信する際の IP アドレスです。

Supported Node Names

クォーラムサーバーがサポートする各クラスターノード名 (39 文字以下)。このノード名は、クォーラムサーバーのプロセスを実行しているシステム上の `qs_authfile` に入れます。

ボリュームマネージャーのプランニング

LVM を使用するディスクレイアウトを設計する際は、以下の点を考慮してください。

- 高可用性のアプリケーション、サービス、またはデータを含むボリュームグループは、一次ノードとすべての引き継ぎノードから使用可能な 1 つまたは複数のバス上にある必要があります。
- 高可用性のアプリケーション、サービス、およびデータは、高可用性でないアプリケーション、サービス、およびデータとは別のボリュームグループに配置する必要があります。
- 高可用性のアプリケーション、サービス、およびデータのうち、同時に制御を移行する必要があるものは、同じ単一ボリュームグループまたは複数のボリュームグループ上にグループ化する必要があります。
- 2 つの異なる高可用性のアプリケーション、サービス、またはデータの制御を別々に移行する必要がある場合は、同じボリュームグループにグループ化してはなりません。
- ルートディスクは別のノード上でアクティブ化されるボリュームグループに属してはなりません。

ボリュームグループと物理ボリュームのワークシート

高可用性アプリケーションで使う各ボリュームグループの構築にどの物理ディスク、LUN、またはディスクアレイグループを使うかを指定することで、物理ディスク構成を編成し記録することができます。ボリュームグループと物理ボリュームのワークシート([265 ページ](#))を使ってください。

注記: デフォルトのボリュームグループ名 (vg01、vg02 など) 以外のボリュームグループ名を使用することをお勧めします。そのボリュームグループに関連するハイアベイラビリティアプリケーションを示すボリュームグループ名 (たとえば、/dev/vgdatabase) を選択すると、クラスタの管理が簡単になります。

クラスタ構成のプランニング

クラスタは、障害から最も迅速に回復できるように設計しなければなりません。実際に障害から回復するのに必要な時間は、以下の要因により異なります。

- `MEMBER_TIMEOUT` の長さ。推奨値については、「[クラスタ構成のパラメーター](#)」に記載されたこのパラメーターの説明を参照してください。
- パッケージ制御スクリプトの起動/停止命令の設計。これらの命令は、高速実行されるように記述する必要があります。
- アプリケーションとデータベースの回復時間。これらの時間は最短になるよう設計されなくてはなりません。

また、クラスタ全体の一貫性を維持するため、以下の項目に従ってください。

- ユーザー名は、すべてのノードで同じものを使用する。
- UID は、すべてのノードで同じものを使用する。
- GID は、すべてのノードで同じものを使用する。
- システム領域のアプリケーションは、すべてのノードで同じものを使用する。
- システム時刻をクラスタ全体で合わせる。
- /usr または /opt のファイルなど、複数のノードが使用する可能性のあるファイルは、すべてのノードで同じものを使用する。

ハートビートサブネットとクラスタの再編成時間

クラスタの再編成時間は、ハートビートサブネットの数によって決まります。

クラスタに単一のハートビートネットワークしかなく、そのネットワーク上のネットワークカードで障害が発生すると、障害が検出されて IP アドレスがスタンバイインターフェイスに切り替えられている間、ハートビートは失われます。クラスタは失われたハートビートを障害として処理し、そのノードを除いて再編成しようとします。このような問題を回避するために、単一のハートビートネットワークを持つクラスタの場合、`MEMBER_TIMEOUT` の値を 14 秒以上にする必要があります。

複数のハートビートサブネットが存在し、いずれかに障害が発生した場合は、ハートビートは別のサブネットに移るため、`MEMBER_TIMEOUT` の値を小さく設定できます。

注記: ハートビートの構成要件については、この章で後述する `HEARTBEAT_IP` パラメーターについての説明を参照してください。クラスタ再編成時間の管理についての詳細は、`MEMBER_TIMEOUT` パラメーターの説明と、「[ノードがタイムアウトしたときの動作](#)」(68 ページ)に記載されている詳しい説明を参照してください。トラブルシューティングについては、「[MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスタの再編成](#)」(244 ページ)を参照してください。

ホスト名アドレスファミリについて: IPv4 専用、IPv6 専用、および混合モード

Serviceguard は、ノードのホスト名 (および存在する場合にはクォラムサーバーのホスト名) をネットワークアドレスファミリに解決するための次の 3 つのモードをサポートしています。

- IPv4 専用
- IPv6 専用
- 混合

IPv4 専用では、Serviceguard はホスト名を IPv4 アドレスだけで解決しようと試みます。

- ① **重要:** IPv4 専用、IPv6 専用、または混合のどのモードでも、IPv6 ハートビート、または定常 IP アドレスや再配置可能 IP アドレスを構成できます。IPv4 専用または混合モードでは、IPv4 ハートビート、または定常 IP アドレスや再配置可能 IP アドレスを構成できます。

IPv6 専用では、Serviceguard はホスト名を IPv6 アドレスだけで解決しようと試みます。

混合では、ホスト名を解決するときに、Serviceguard は IPv4 と IPv6 の両方のアドレスファミリーで解決しようと試みます。

クラスターが使用するアドレスファミリーは、クラスター構成ファイルの中で (`HOSTNAME_ADDRESS_FAMILY` を `IPV4`、`IPV6`、または `ANY` に設定する)、または `cmquerycl` (1m) の `-a` オプションを使用して指定します。「[クラスターホスト名のアドレスファミリーの指定](#)」(142 ページ) を参照してください。デフォルトは、`IPV4` です。詳細と重要な規則および制限事項については、以降の項を参照してください。

IPv4 専用モードについて

IPv4 はデフォルトモードです。`IPV6` または `ANY` を指定しない限り (クラスター構成ファイル内、または `cmquerycl -a` を介して)、Serviceguard は常に、ノードのホスト名 (および存在する場合はクォーラムサーバーのホスト名) を IPv4 アドレスで解決しようと試み、IPv6 アドレスでの解決は試みません。つまり、各ホスト名が 1 つ以上の IPv4 アドレスで解決できることを確認する必要があります。

注記: これは、ホスト名解決にだけあてはまります。`HOSTNAME_ADDRESS_FAMILY` パラメーターが何に設定されているかに関係なく、IPv6 ハートビートとデータ LAN は指定できます (IPv4 ハートビートとデータ LAN は IPv4 モードと混合モードでのみ指定可能です)。

IPv6 専用モードについて

IPv6 専用モードを構成している場合 (`HOSTNAME_ADDRESS_FAMILY` を `IPV6` に設定するか、`cmquerycl -a ipv6` を実行する)、ハートビートおよび定常 IP アドレスと再配置可能 IP アドレス、クォーラムサーバーのアドレス (存在する場合) も含め、クラスターが使用するホスト名とアドレスはすべて IPv6 アドレスであるか、IPv6 アドレスで解決する必要があります。唯一の例外は、各ノードの IPv4 ループバックアドレスです。このアドレスは、`/etc/hosts` から削除できません。

注記: IPv6 専用クラスターアプリケーションのクライアントがホスト名解決をどのように処理するかは、システム管理者またはネットワーク管理者の裁量事項です。これに関する HP の要件および推奨事項はありません。

IPv6 専用モードでは、通常、すべての Serviceguard デーモンがノード間通信に IPv6 アドレスを使用します。ただし、ローカル (イントラノード) 通信は IPv4 ループバックアドレス上で発生することがあります。

IPv6 についての詳細は、[付録 D \(268 ページ\)](#) を参照してください。

IPv6 専用モードの規則と制限事項

- ① **重要:** これらの最新情報およびその他の制限事項の詳細は、最新版の Serviceguard for Linux リリースノートを参照してください。

- Red Hat 5 および Red Hat 6 クラスターはサポートされません。

注記: これは、`HOSTNAME_ADDRESS_FAMILY` が `ANY` に設定されている場合も同じです。Red Hat 5 は IPv4 のみのクラスターだけをサポートします。

- クラスターによって使用されるアドレスはすべて、各ノードの `/etc/hosts` ファイルに含まれている必要があります。また、このファイルには以下のエントリーが必要です。
`:::1 localhost ipv6-localhost ipv6-loopback`
 ホスト名解決についての詳細および推奨事項については、「名前解決の構成」(124 ページ)を参照してください。
- ノードの IPv4 ループバックアドレス以外のすべてのアドレスは、IPv6 でなければなりません。ノードの IPv4 ループバックアドレスは、`/etc/hosts` から削除できません。
- ノードのパブリック LAN アドレス (外部ネットワークでのこのノードのアドレス) は、`/etc/hosts` 内の最後に記述する必要があります。
 そうしなければ、そのアドレスがクラスターに構成されていない場合でも使用されてしまう可能性があります。
- `~/.rhosts` や `/etc/hosts.equiv` ではなく、`$SGCONF/cmclnodelist` を使って、未構成ノードへのルートアクセスを許可する必要があります。

注記: これは、`HOSTNAME_ADDRESS_FAMILY` が `ANY` に設定されている場合にも該当します。詳細は、「未構成ノードへのルートアクセスの許可」(123 ページ)を参照してください。

- コーラムサーバーを使用している場合は、コーラムサーバーのホスト名 (および、存在する場合は、`QS_ADDR` によって指定された代替コーラムサーバーのアドレス) が IPv6 アドレスで解決されるようにする必要があります。また、バージョン A.04.00 以降を使用する必要があります。詳細は、最新の Quorum Server のリリースノートを参照してください。これは、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にあります。

注記: コーラムサーバー自体が IPv6 専用システムである場合があります。その場合、IPv6 専用モードと混合モードでクラスターを稼働させることができますが、IPv4 専用モードではクラスターを稼働させることはできません。

- コーラムサーバーを使い、そのコーラムサーバーがクラスターとは異なるサブネット上にある場合、IPv6 対応ルーターを使う必要があります。
- IPv6 アドレスの場合、ホスト名のエイリアスは、オペレーティングシステムの制限があるためサポートされません。

注記: これは、`HOSTNAME_ADDRESS_FAMILY` が `IPV6` または `ANY` のどちらかに設定されているかにかかわらず、すべての IPv6 アドレスにあてはまります。

- クロスサブネット構成は、IPv6 専用モードではサポートされていません。
- 仮想マシンはサポートされません。
`HOSTNAME_ADDRESS_FAMILY` が `ANY` または `IPV6` に設定されている場合、仮想マシンをノードまたはパッケージとして保持することはできません。

IPv6 専用モードの推奨事項

- ① **重要:** 最新の手順と推奨事項は、最新の Serviceguard for Linux のリリースノートを参照してください。

- クラスターを IPv6 専用モードに移行することに決定した場合は、クラスターがダウンしている間に移行する必要があります。

混合モードについて

混合モードを構成 (`HOSTNAME_ADDRESS_FAMILY` を `ANY` に設定、または `cmquerycl -a any`) している場合は、ハートビート、クォーラムサーバーのアドレス (存在する場合) を含め、クラスターによって使用されるアドレスは IPv4 または IPv6 のいずれかのアドレスになります。Serviceguard は、最初にノードのホスト名の IPv4 アドレスでの解決を試み、それに失敗した場合に、IPv6 アドレスでの解決を試みます。

混合モードの規則と制限事項

- ❶ **重要:** これらの最新情報およびその他の制限事項の詳細は、最新版の Serviceguard リリースノートを参照してください。

- Red Hat 5 および Red Hat 6 クラスターはサポートされません。

注記: これは、`HOSTNAME_ADDRESS_FAMILY` が IPv6 に設定されている場合も同じです。Red Hat 5 は IPv4 のみのクラスターだけをサポートします。

- 各ノードのホスト名解決ファイル (たとえば `/etc/hosts`) には、クラスター全体で使用されているすべての IPv4 および IPv6 アドレスに対するエントリーが含まれている必要があります (他のプライベートアドレス同様に、すべての `STATIONARY_IP` アドレスと `HEARTBEAT_IP` アドレスが必要です)。このファイルには必ず 1 つ以上の IPv4 アドレスが存在します (`/etc/hosts` の場合、IPv4 ループバックアドレスは削除できません)。また、このファイルには以下のエントリーが必要です。

```
::1 localhost ipv6-localhost ipv6-loopback
```

ホスト名解決についての詳細および推奨事項については、「[名前解決の構成](#)」(124 ページ)を参照してください。

- `~/.rhosts` や `/etc/hosts.equiv` ではなく、`$SGCONF/cmclnodelist` を使って、未構成ノードへのルートアクセスを許可する必要があります。
詳細は、「[未構成ノードへのルートアクセスの許可](#)」(123 ページ)を参照してください。
- IPv6 アドレスの場合、ホスト名のエイリアスは、オペレーティングシステムの制限があるためサポートされません。

注記: これは、`HOSTNAME_ADDRESS_FAMILY` が `IPV6` または `ANY` のどちらかに設定されているかにかかわらず、すべての IPv6 アドレスにあてはまります。

- クロスサブネット構成はサポートされていません。
これは、`HOSTNAME_ADDRESS_FAMILY` が `IPV6` に設定されている場合にも該当します。
この構成についての詳細は、「[クロスサブネット構成](#)」(25 ページ)を参照してください。
- 仮想マシンはサポートされません。

`HOSTNAME_ADDRESS_FAMILY` が `ANY` または `IPV6` に設定されている場合、仮想マシンをノードまたはパッケージとして保持することはできません。

クラスター構成のパラメーター

一連のクラスターパラメーターを定義する必要があります。これらのパラメーターは、クラスター内のすべてのノードに配布される、バイナリ形式クラスター構成ファイルに保存されます。これらのパラメーターは、クラスター構成のテンプレートファイルを編集することにより構成できます。テンプレートファイルは `cmquerycl` コマンドを使って作成できます。詳しくは、「[クラスターの構成](#)」(141 ページ)を参照してください。

注記: クラスターの稼働中に行える変更の概要については、「[クラスターの再構成](#)」(209 ページ)を参照してください。

以下のパラメーターを構成しなければなりません。

CLUSTER_NAME

cmviewcl コマンドや他のコマンドの出力に表示されるクラスターの名前。クラスター構成ファイルにもクラスターの名前が記述されています。

クラスター名には、スペース、スラッシュ (/)、バックスラッシュ (\)、アスタリスク (*) を使用しないでください。

注記: また、クォーラムサーバーを使っている場合は、クラスター名には、アットマーク (@)、等号 (=)、論理和記号 (|)、セミコロン (;) を使用しないでください。

これらの文字は非推奨となりました。Quorum Server を利用しない場合でも使わないようにしてください。

その他の文字は使用できます。クラスター名に使用できる文字数は、最大 39 文字です。

- △ **注意:** クラスター名が、クラスターノードを構成しているサブネット内で一意であることを確認してください。場合によっては Serviceguard が名前の重複を検出できず、予期しない問題が発生することがあります。

特に同じ名前の 2 つのクラスターに対して同じクォーラムサーバーを使用しないでください。使用すると、いずれかのクラスターは必要時にクォーラムサーバーのアービトレーションサービスを受けられず、その結果、再編成に失敗する可能性があります。

HOSTNAME_ADDRESS_FAMILY

Serviceguard がクラスターノード名とクォーラムサーバーホスト名の解決に必要なインターネットプロトコルアドレスファミリを指定します。有効な値は、IPV4、IPV6 および ANY です。デフォルトは、IPV4 です。

- IPV4 を指定すると、Serviceguard は名前を IPv4 アドレスで解決しようと試みます。
- IPV6 を指定すると、Serviceguard は名前を IPv6 アドレスで解決しようと試みます。
- ANY を指定すると、Serviceguard は名前を IPv4 および IPv6 アドレスの両方で解決しようと試みます。

- ① **重要:** 重要な情報については、「[ホスト名アドレスファミリについて: IPv4 専用、IPv6 専用、および混合モード](#)」(79 ページ)を参照してください。<http://www.hp.com/go/linux-serviceguard-docs>にある最新の Serviceguard のリリースノートも参照してください。

QS_HOST

クォーラムサーバー機能を提供し、現在のクラスター外にあるシステムの完全なホスト名または IP アドレス。Red Hat 5 では、IPv4 アドレスである (または IPv4 アドレスに解決できる) 必要があります。SLES 11 では、HOSTNAME_ADDRESS_FAMILY が ANY に設定されている場合は、IPv4 または IPv6 アドレスのどちらにも指定 (解決) できます。ただし、ANY 以外の場合は、HOSTNAME_ADDRESS_FAMILY の設定に一致している必要があります。このパラメーターは、クラスター内でクォーラムサーバーをタイプブレイクサービスとして利用する場合にのみ使います。また、代替アドレス (QS_ADDR) を指定することもできます。クラスターノードは、これを使ってクォーラムサーバーにアクセスできます。

詳細は、「[クラスターロックのプランニング](#)」(77 ページ)と「[クォーラムサーバーの指定](#)」(143 ページ)を参照してください。<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software]にある最新版の『HP Serviceguard Quorum Server Version A.04.00 Release Notes』の「Configuring Serviceguard to Use the Quorum Server」も参照してください。

- ① **重要:** また、IPv6 専用サーバーの要件と制限事項についての重要な情報は、「[ホスト名アドレスファミリについて: IPv4 専用、IPv6 専用、および混合モード](#)」(79 ページ)を参照してください。

このパラメーターはクラスターの稼働中に変更することもできます。重要な情報は、「[オンラインでクォーラム構成を変更したときの動作](#)」(37 ページ)を参照してください。

QS_ADDR

代替となるクォーラムサーバーの完全なホスト名または IP アドレス。Red Hat 5 および Red Hat 6 では、IPv4 アドレスである (または IPv4 アドレスに解決できる) 必要があります。SLES 11 では、`HOSTNAME_ADDRESS_FAMILY` が ANY に設定されている場合は、IPv4 または IPv6 アドレスのどちらにも指定 (解決) できます。ただし、ANY 以外の場合は、`HOSTNAME_ADDRESS_FAMILY` の設定に一致している必要があります。このパラメーターは、クォーラムサーバーを使っており、アクセス可能な代替サブネット上のアドレスを指定する場合にのみ使います。SLES 11 では、`HOSTNAME_ADDRESS_FAMILY` が ANY に設定されている場合、代替サブネットが `QS_HOST` と同じアドレスファミリーを使用する必要はありません。詳細は、「[クラスターロックのプランニング](#)」(77 ページ)と「[クォーラムサーバーの指定](#)」(143 ページ)を参照してください。

- ❗ **重要:** 個々の Serviceguard バージョンとクォーラムサーバーに適用される個別の作業手順は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にある最新版の『HP Serviceguard Quorum Server Version A.04.00 Release Notes』の「Configuring Serviceguard to Use the Quorum Server」を参照してください。

このパラメーターはクラスターの稼働中に変更することもできます。重要な情報は、「[オンラインでクォーラム構成を変更したときの動作](#)」(37 ページ)を参照してください。

QS_POLLING_INTERVAL

クォーラムサーバーが動作していることを確認するために連絡をとる間隔 (マイクロ秒)。デフォルトは 300,000,000 マイクロ秒 (5 分) です。最小値は 10,000,000 (10 秒) です。最大値は 2,147,483,647 (約 35 分) です。

このパラメーターはクラスターの稼働中に変更することもできます。重要な情報は、「[オンラインでクォーラム構成を変更したときの動作](#)」(37 ページ)を参照してください。

QS_TIMEOUT_EXTENSION

`QS_TIMEOUT_EXTENSION` を使用すると、クォーラムサーバーへの現在の接続 (または接続の試み) が失敗したとみなされるまでの間隔を長くすることができます。ただし、その前に『HP Serviceguard Quorum Server Version A.04.00 Release Notes』、および特にその中の「About the QS Polling Interval and Timeout Extension」、「Network Recommendations」、および「Setting Quorum Server Parameters in the Cluster Configuration File」の項を参照してください。

このパラメーターはクラスターの稼働中に変更することもできます。重要な情報は、「[オンラインでクォーラム構成を変更したときの動作](#)」(37 ページ)を参照してください。

NODE_NAME

クラスター内のノードとなる各システムのホスト名。

- ⚠ **注意:** クラスターノード上に構成されているサブネット内では、ノード名を一意的な名前にしてください。状況によっては、Serviceguard が重複する名前を検出できず、予期しない問題を起こす場合があります。

完全ドメイン名を使うことはできません。たとえば、`ftsys9.cup.hp.com` ではなく、`ftsys9` と記入します。クラスターは、最大 16 ノードで構成することができます。

- ❗ **重要:** ノード名は、39 文字以下でなければならず、大文字小文字の区別があります。各ノードでは、クラスター構成ファイルでの `node_name` は、パッケージ構成ファイル内の対応する `node_name` と正確に一致しなければなりません (第6章「[パッケージとサービスの構成](#)」(157 ページ)を参照)。またこれらの名前は、ノードのネットワーク構成で指定された名前の `hostname` 部分と正確に一致しなければなりません (上記の例では、`ftsys9` は、クラスター構成ファイルとパッケージ構成ファイルでは完全にこの形式でなければならず、DNS データベースでは `ftsys9.cup.hp.com` となっていなければなりません)。

このリストで `NODE_NAME` の直後に続くパラメーター (`NETWORK_INTERFACE`、`HEARTBEAT_IP`、`STATIONARY_IP`、`CLUSTER_LOCK_LUN`、`CAPACITY_NAME`、`CAPACITY_VALUE`) は、直前の `NODE_NAME` エントリーによって指定されたノードにのみ適用されます。

`CLUSTER_LOCK_LUN`

各ノード上でロック LUN として使うデバイスファイルのパス名。パス名の最大文字数は 39 文字です。

「[ロック LUN の設定](#)」(130 ページ)と「[ロック LUN の指定](#)」(143 ページ)を参照してください。

このパラメーターはクラスタの稼働中に変更することもできます。「[オンラインでのクラスタロック LUN 構成のアップデート](#)」(217 ページ)を参照してください。重要な情報については、「[オンラインでクォーラム構成を変更したときの動作](#)」(37 ページ)も参照してください。

`NETWORK_INTERFACE`

先行する `NODE_NAME` で指定されたノードの、ハートビートまたはユーザーデータに使用する各 LAN の名前。たとえば `eth0` などです。「[ホスト名アドレスファミリーについて: IPv4 専用、IPv6 専用、および混合モード](#)」(79 ページ)、`HEARTBEAT_IP`、`STATIONARY_IP` も参照してください。

注記: IP 監視目的のために `SUBNET` としてこのクラスタ構成ファイルで構成されているサブネット、またはパッケージ構成ファイルで `monitored_subnet` として構成されているサブネット (あるいは従来のパッケージでは `SUBNET`、「[パッケージ構成のプランニング](#)」(92 ページ)を参照) は、`NETWORK_INTERFACE` と `STATIONARY_IP` か `HEARTBEAT_IP` のいずれかを使用してクラスタ構成ファイルで指定する必要があります。同様に、再配置可能アドレス用のパッケージで使用されるサブネットも、`NETWORK_INTERFACE` と `STATIONARY_IP` か `HEARTBEAT_IP` のいずれかを使用してクラスタ内に構成する必要があります。再配置可能アドレスの詳細は、「[定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット](#)」(54 ページ)と、パッケージの `ip_` パラメーター(170 ページ)の説明を参照してください。

オンラインでの構成変更についての詳細は、「[稼働中のクラスタのクラスタネットワーク構成の変更](#)」(214 ページ)を参照してください。

`HEARTBEAT_IP`

クラスタのハートビートを搬送するサブネットに、このノードが接続していることを示す IP アドレス。

注記: IP 監視目的のために `SUBNET` としてこのクラスタ構成ファイルで構成されているサブネット、またはパッケージ構成ファイルで `monitored_subnet` として構成されているサブネット (あるいは従来のパッケージでは `SUBNET`、「[パッケージ構成のプランニング](#)」(92 ページ)を参照) は、`NETWORK_INTERFACE` と `STATIONARY_IP` か `HEARTBEAT_IP` のいずれかを使用してクラスタ構成ファイルで指定する必要があります。同様に、パッケージによって再配置可能アドレス用に使われるサブネットも、`NETWORK_INTERFACE` と、`STATIONARY_IP` または `HEARTBEAT_IP` のいずれかを通じてクラスタ内に構成しなければなりません。再配置可能アドレスの詳細は、「[定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット](#)」(54 ページ)と、パッケージの `ip_` パラメーター(170 ページ)の説明を参照してください。

`HOSTNAME_ADDRESS_FAMILY` が `IPv4` または `ANY` に設定されている場合、ハートビート IP アドレスは、以下に示す例外を除き、IPv4 アドレスまたは IPv6 アドレスで指定できます。`HOSTNAME_ADDRESS_FAMILY` が `IPv6` に設定されている場合、ハートビート IP アドレスはすべて IPv6 アドレスで指定する必要があります。

IPv6 のアドレス形式の詳細は、「[IPv6 アドレスの種類](#)」(268 ページ)を参照してください。特定のサブネット上のハートビート IP アドレスは、すべて同じタイプでなければなりません。サイトローカルでは IPv4 または IPv6、グローバルでは IPv6 です。

オンラインでの構成変更についての詳細は、「[稼働中のクラスターのクラスターネットワーク構成の変更](#)」(214 ページ)を参照してください。

ハートビート構成の要件:

以下のいずれかの最小構成を使い、ハートビート用に必ず 2 つ以上のネットワークインターフェイスが必要です。

- ハートビートサブネット 2 つ。

または

- 2 つのスレーブを備えた高可用性モード (モード 1) のボンディングを使ったハートビートサブネット 1 つ。

1 つのインターフェイスに対して複数のハートビート IP アドレスを構成することはできません。各 `NETWORK_INTERFACE` に対して構成可能な `HEARTBEAT_IP` は 1 つのみです。

注記: Serviceguard のコマンド `cmapplyconf`、`cmcheckconf`、`cmquerycl` はこれらの最小限の要件を満たしているかどうかを確認し、最低限のネットワークレベルが満たされていない場合は警告を表示します。この警告が表示された場合は、ネットワーク構成全体で要件が満たされているかどうかを確認する必要があります。

仮想マシングストをノードとして使用している場合、ゲスト上に 1 つのハートビートネットワークがあり、そのネットワークを、前述の 2 番目に示した NIC ボンディング (VMware ESX Server) を使用するネットワークがバックアップしていれば、構成が有効になります (警告を無視できます)。

クロスサブネットに関する留意事項

ハートビートパスの IP アドレスは通常、各ノード上で同じサブネットにあります。サブネットがルーターで結合されていれば、一部のノードについてはあるサブネットからハートビートを搬送し、別のノードについては別のサブネットからハートビートを搬送するように、複数のサブネット上にハートビートを構成することができます。

これを、クロスサブネット構成といいます。この場合、**少なくとも 2 つのハートビートパス**を各クラスターノードに構成しなければなりません。各ノードのそれぞれのハートビートサブネットでは、別ノードのハートビートサブネットに対して物理的に個別の経路指定が行われます (つまり、各ハートビートパスは物理的に分離されています)。「[クロスサブネット構成](#)」(25 ページ)を参照してください。

注記: IPv6 ハートビートサブネットは、クロスサブネット構成ではサポートされません。

注記: リモートプロシージャール (RPC) プロトコルおよびサービスを使用する場合は、プライベートなハートビートネットワークを使用することはお勧めできません。RPC では、各ネットワークアダプターデバイスや I/O カードが経路指定可能なネットワークに接続されていることを想定しています。独立した、またはプライベートなハートビート LAN は経路指定できず、RPC 要求/応答がこの LAN に渡されると、処理されることなくタイムアウトとなる可能性があります。

NFS、NIS、NIS+、CDE は、よく使用される RPC ベースのアプリケーションの例です。その他のサードパーティー製または自家製のアプリケーションも RPC API ライブラリ経由で RPC サービスを使っているかもしれません。必要であれば、アプリケーションベンダーにお問い合わせください。

STATIONARY_IP

各サブネット上のノードの IP アドレスで、クラスターのハートビートを搬送しませんが、パッケージで監視することは可能です。

注記: IP 監視目的のために `SUBNET` としてこのクラスター構成ファイルで構成されているサブネット、またはパッケージ構成ファイルで `monitored_subnet` として構成されているサブネット (あるいは**従来の**パッケージでは `SUBNET`、[「パッケージ構成のプランニング」](#) (92 ページ) を参照) は、`NETWORK_INTERFACE` と `STATIONARY_IP` か `HEARTBEAT_IP` のいずれかを使用してクラスター構成ファイルで指定する必要があります。同様に、パッケージによって再配置可能アドレス用に使われるサブネットも、`NETWORK_INTERFACE` と、`STATIONARY_IP` または `HEARTBEAT_IP` のいずれかを通じてクラスターに構成しなければなりません。再配置可能アドレスの詳細は、[「定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット」](#) (54 ページ) と、パッケージの `ip_` パラメーター([170 ページ](#)) の説明を参照してください。

`HOSTNAME_ADDRESS_FAMILY` が IPv4 または ANY に設定されている場合、定常 IP アドレスは、以下に示す例外を除き、IPv4 アドレスまたは IPv6 アドレスのいずれかで指定できます。`HOSTNAME_ADDRESS_FAMILY` が IPv6 に設定されている場合、クラスターによって使用される IP アドレスはすべて IPv6 アドレスで指定する必要があります。

アプリケーションデータとハートビートメッセージを分けたい場合、ここに監視対象の非ハートビートサブネットを 1 つ以上定義します。監視対象のサブネット数は任意に設定できます。

定常 IP アドレスは、IPv4 アドレスまたは IPv6 アドレスで指定できます。IPv6 アドレスの詳細は、[「IPv6 アドレスの種類」](#) (268 ページ) を参照してください。

オンラインでの構成変更についての詳細は、[「稼働中のクラスターのクラスターネットワーク構成の変更」](#) (214 ページ) を参照してください。

`CAPACITY_NAME`、`CAPACITY_VALUE`

ノードキャパシティのパラメーター。`CAPACITY_NAME` と `CAPACITY_VALUE` のパラメーターを使用して、ノードキャパシティを定義します。ノードキャパシティはパッケージ重みに対応しています。ノードキャパシティと対応するパッケージ重みとが比較され、ノードでパッケージを実行できるかどうかが決まります。

`CAPACITY_NAME` 名は、先頭と末尾が英数字で、その他は英数字、ドット (.)、ダッシュ (-)、下線 (_) のみの任意の文字列を指定できます。指定できる文字数は、最大 39 文字です。`CAPACITY_NAME` は、クラスター内で一意である必要があります。

`CAPACITY_VALUE` は、先行する `CAPACITY_NAME` の値を指定します。0~1000000 までの、浮動小数点値でなければなりません。キャパシティの値は、少なくとも Serviceguard については任意です。対応するパッケージとの関係でのみ意味があります。

キャパシティの定義はオプションですが、`CAPACITY_NAME` を指定した場合は、`CAPACITY_VALUE` も指定する必要があります。`CAPACITY_NAME` を先に指定してください。

注記: ノードキャパシティが定義されていて、パッケージの [failover_policy](#)(166 ページ) が `min_package_node` であるか、または [failback_policy](#)(167 ページ) が `automatic` の場合、`cmapplyconf` は失敗します。

1 つのノードに複数のキャパシティを指定するには、各キャパシティに対してこれらのパラメーターを繰り返し指定します。1 クラスターにつき最大 4 つまでキャパシティを指定できます。ただし、予約済みの `CAPACITY_NAME` である `package_limit` を使用する場合は、クラスター全体でそのキャパシティしか使用できません。

`package_limit` 以外のすべてのキャパシティでは、すべてのパッケージのデフォルトの重みは 0 です。しかし、`package_limit` 以外のどのキャパシティについても異なるデフォルトの重みを指定できます。このリストの後半にある `WEIGHT_NAME` と `WEIGHT_DEFAULT` のエントリを参照してください。

詳細は、[「パッケージの重みについて」](#) (108 ページ) を参照してください。

このパラメーターはクラスターの稼働中に変更することもできます。変更によって実行中のパッケージに障害が発生する場合は警告が生成されます。

MEMBER_TIMEOUT

Serviceguard が、ノードに障害が発生したと判断し、このノードを除いてクラスターの再編成を開始するまでの時間 (マイクロ秒単位)。

デフォルト値: 14 秒 (14,000,000 マイクロ秒)。

フォーラムサーバーもしくは FibreChannel クラスターロックを使用しているか、クラスターロックをまったく使用していない場合、このデフォルト値の場合、フェイルオーバー時間は約 18~22 秒となります。値を 25 秒に増やすと、フェイルオーバー時間は約 29~39 秒となります。SCSI クラスターロックまたはデュアル Fibre Channel クラスターロックを使用している場合、この時間は 5~13 秒程度長くなります。

サポートされている最大値: 300 秒 (300,000,000 マイクロ秒)。

60 秒 (60,000,000 マイクロ秒) より大きい値を入力する場合、cmcheckconf と cmmapplyconf は大きな値を使用しようとしている事実を確認します。

サポートされている最小値:

- ハートビートサブネットが複数あるクラスターの場合 3 秒。
- 1 つのハートビート LAN しか持たないクラスターの場合、14 秒

サポートされている最小値の 3 秒を使用した場合、4~5 秒のフェイルオーバー時間を達成できます。

注記: ここに示したフェイルオーバーの概算時間は、フェイルオーバーのうち Serviceguard 構成要素に適用されるものです。言い換えると、パッケージはこの時間内に起動され、引き継ぎノード上で稼働状態になると予想できますが、パッケージが実行しているアプリケーション自体は起動までにもう少し時間がかかる可能性があります。

ロック LUN を使用するクラスターでは、ほとんどの場合、MEMBER_TIMEOUT の最小値である 14 秒が適切です。

14 秒未満の MEMBER_TIMEOUT 値を使用する大半のクラスターでは、ロック LUN よりフォーラムサーバーの方が適しています。ディスクロックを取得するまでに、MEMBER_TIMEOUT の 0.2 倍を超える時間がかかった場合、クラスターは失敗します。つまり、ディスクベースのフォーラムデバイス (ロック LUN) を使用している場合は、クラスター内のノード、ディスクへの接続、およびディスク自体が、MEMBER_TIMEOUT の 0.2 倍の時間以内に 10 回のディスク書き込みを実行できるほど素早く応答できることを確認する必要があります。

値の設定方法を決定するにあたっては、以下のガイドラインに留意してください。

ガイドライン: クラスターの再編成の発生回数を少なくすべきか (ただし再編成の動作は遅い)、再編成の動作を速くすべきか (ただし発生回数は多くなる)、どちらが重要かを判断する必要があります。

- クラスターの再編成の速度を最大にするには、お使いのクラスターに適用できる最小値を使います。ただし、この設定では、システムのハングやネットワーク負荷の急上昇によって MEMBER_TIMEOUT 値で指定された時間内にハートビート信号を送信できない場合、クラスターの再編成が行われ、ノードがクラスターから削除されてリブートされる点に注意してください。複数のノードが影響を受けることがあります。たとえば、ブロードキャストストームなどのネットワークイベントが起きると、何台かまたはすべてのノードで、パケットを処理している間はカーネル割り込みが無効となり、ハートビートメッセージの送信と処理ができなくなります。

トラブルシューティングについての情報は、「**MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスターの再編成**」(244 ページ)を参照してください。

- 再編成を少なくするには、10～25 秒 (10,000,000～25,000,000 マイクロ秒) の範囲の設定を使用します。ただし、デフォルト値より大きい値を使用すると、デフォルト値の場合よりも再編成が遅くなる点に注意してください。この範囲の値は、ほとんどのインストール環境に適しています。

「ノードがタイムアウトしたときの動作」(68 ページ)、「クラスターデーモン: cmcld」(30 ページ)、および <http://www.hp.com/go/linux-serviceguard-docs> -> [White papers] にあるホワイトペーパー『Optimizing Failover Time in a Serviceguard Environment (version A.11.19 and later)』も参照してください。

このパラメーターはクラスターの稼働中に変更することもできます。

AUTO_START_TIMEOUT

クラスターの自動起動中に、ノードがクラスターに結合する試みを中止するまでに待つ時間です。クラスターが起動を完了する前に、すべてのノードは、他のノードが起動を開始するのを、このパラメーターで指定した時間だけ待機します。この待機時間は、クラスター内の最も長い起動時間に基づいて選択する必要があります。最も長い起動時間から最も短い起動時間を引いた値に 600 秒 (10 分) を加えた値を入力します。

デフォルトは、600,000,000 マイクロ秒です。

このパラメーターはクラスターの稼働中に変更することもできます。

NETWORK_POLLING_INTERVAL

Serviceguard が定期的にすべての LAN インターフェイスをポーリングする間隔を指定します (リンクレベル、および IP MONITOR 用に構成されているインターフェイス)。

デフォルト値は 2,000,000 マイクロ秒 (2 秒) です。したがって、2 秒ごとにネットワークマネージャーが各ネットワークインターフェイスをポーリングして、情報を送受信できることを確かめます。

サポートされている最小値は 1,000,000 (1 秒)、最大値は 30 秒です。

次に例を示します。

- `NETWORK_POLLING_INTERVAL` が 6,000,000 (6 秒) と定義されている場合、ポーリングは 6 秒目、12 秒目、と行われていきます。

`NETWORK_POLLING_INTERVAL` が 9,000,000 (9 秒) と定義されている場合、ポーリングは 9 秒目、18 秒目、と行われていきます。

- Serviceguard は、各 LAN インターフェイスが連続して失敗または受信したパケットの数を算出することにより LAN インターフェイスが DOWN か UP かを判断するときにもこのパラメーターを使用します。

インターフェイスが IP レベルで監視されている場合、`NETWORK_POLLING_INTERVAL` を 8 秒以上に設定すると、各 LAN インターフェイスが DOWN または UP とみなされるまでに連続して失敗または受信できるパケットの数は 2 です。

次に例を示します。

`NETWORK_POLLING_INTERVAL` が 10 と定義されている場合、インターフェイスの IP レベルでの障害および回復は 25 以内に検出されます。

以下に、IP 監視イーサネットインターフェイスについて、ネットワークポーリング間隔 (NPI) の値によって障害および回復の検出時間がどのように変わるのかを示します。

表 5 IP 監視イーサネットインターフェイスにおける障害および回復の検出時間

ネットワークポーリング間隔 (NPI) の値 (秒)	障害および回復の検出時間 (秒)
1	NPI の 8 倍 ～ NPI の 9 倍以内
2	NPI の 4 倍 ～ NPI の 5 倍以内

表 5 IP 監視イーサネットインターフェイスにおける障害および回復の検出時間 (続き)

ネットワークポーリング 間隔 (NPI) の値 (秒)	障害および回復の検出時間 (秒)
3	NPI の 3 倍 ~ NPI の 4 倍以内
>=4	NPI の 2 倍 ~ NPI の 3 倍以内

- ① **重要:** デフォルト値を使用することを強くお勧めします。この値を変更すると、リンクレベルおよび IP レベルの監視によるネットワーク障害の検出時間に影響を与える可能性があります。「[LAN インターフェイスの監視と障害の検出: リンクレベル](#)」(59 ページ) を参照してください。

このパラメーターはクラスターの稼働中に変更することもできます。

CONFIGURED_IO_TIMEOUT_EXTENSION

Serviceguard がノードの障害を検出してから、障害の発生したノード上のすべての保留中の I/O が終了することを確実にするための、待機時間を延長するもので、マイクロ秒単位の数値となります。

このパラメーターは、以下の場合に設定する必要があります。

- iFCP スイッチ間のリンクを介してデータセンターにまたがるソフトウェアミラーリングを使用している遠距離クラスターの場合は、スイッチの `R_A_TOV` の最大値に設定する必要があります。

注記: `CONFIGURED_IO_TIMEOUT_EXTENSION` は、`R_A_TOV` 値を取得できる iFCP スイッチでのみサポートされています。

- NFS マウントされたファイルシステムを使用してパッケージを実行するクラスターノードクライアントと NFS サーバーを接続するスイッチおよびルーターの場合は、「[NFS マウントされたファイルシステムのプランニング](#)」(94 ページ) を参照してください。

`CONFIGURED_IO_TIMEOUT_EXTENSION` の値を設定するには、最初に各スイッチおよびルーターの Maximum Bridge Transit Delay (MBTD) を決定する必要があります。値は、ベンダーから提供されたドキュメントに記載されているはずです。

`CONFIGURED_IO_TIMEOUT_EXTENSION` は、スイッチおよびルーターの**値の合計**に設定してください。NFS サーバーとクラスターノード間に複数のパスが存在する場合は、各パスの値を合計し、最大の数値を使用してください。

- △ **注意:** Serviceguard は、MBTD をサポートするスイッチおよびルーターを介してのみ NFS マウントされたファイルシステムをサポートします。NFS マウントされたファイルシステムを使用している場合、`CONFIGURED_IO_TIMEOUT_EXTENSION` はここでの説明に従って設定する必要があります。

MBTD の詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> にあるホワイトペーパー『Support for NFS as a filesystem type with HP Serviceguard A.11.20 on HP-UX and Linux』を参照してください。

- 上記の両方の条件が当てはまるクラスターの場合、`CONFIGURED_IO_TIMEOUT_EXTENSION` は、前述の 2 つの箇条書きの指示に従い取得した 2 つの**値の大きい数値**を設定してください。

デフォルトは、0 です。

SUBNET

IP Monitoring をオンまたはオフにできるクラスターサブネットの IP アドレス (`IP_MONITOR` を参照してください)。 `NETWORK_INTERFACE` と、 `HEARTBEAT_IP` または `STATIONARY_IP` のいずれかを使って、サブネットをクラスターに構成する必要があります。 `IP_MONITOR`

と *POLLING_TARGET* のすべてのエントリーは、次の *SUBNET* エントリーまでこのサブネットに適用されます。*SUBNET* は、この 3 個セットの先頭に指定する必要があります。デフォルトでは、各クラスターサブネットは *SUBNET* の下にリストされ、サブネットに対して少なくとも 1 つのゲートウェイが検出された場合、*IP_MONITOR* は ON に設定され、*POLLING_TARGET* のエントリーにゲートウェイアドレスが設定されて、ターゲットポーリングが有効になります。ゲートウェイが検出されない場合は、*IP_MONITOR* が OFF でサブネットがリストされます。

詳細は、「[LAN インターフェイスの監視と障害の検出: IP レベル](#)」(59 ページ) を参照してください。

このパラメーターはクラスターの稼働中に変更することもできます。対象となるサブネットがクラスター構成から削除された場合、このパラメーターを付随する *IP_MONITOR* および *POLLING_TARGET* のエントリーとともに削除する必要があります。

IP_MONITOR

先行する *SUBNET* エントリーで指定されたサブネットを IP レイヤーで監視するかどうかを指定します。

サブネットの IP 監視を有効にするには、*IP_MONITOR* を ON に設定します。無効にするには OFF に設定します。

デフォルトでは、対象となる *SUBNET* に対してゲートウェイが検出されると *IP_MONITOR* は ON に設定され、*POLLING_TARGET* のエントリーにゲートウェイアドレスが設定されて、ターゲットポーリングが有効になります。詳細は、以下の *POLLING_TARGET* の説明を参照してください。

第 1 レベルのスイッチを越えた監視を可能にするターゲットポーリングを使うことをお勧めしますが、代わりにピアポーリングを使う場合は、*POLLING_TARGET* を使わずに (すでに入力されている *POLLING_TARGET* エントリーをコメントアウトまたは削除)、この *SUBNET* で *IP_MONITOR* を ON に設定してください。

このサブネットのネットワークインターフェイスに IP レベルで障害が発生し、*IP_MONITOR* が ON に設定されている場合、インターフェイスにはダウン状態を示すマークが付けられます。OFF に設定されている場合、IP レベルでのみ発生した障害は検出されません。

このパラメーターはクラスターの稼働中に変更することもできます。先行する *SUBNET* エントリーが削除された場合は削除しなければなりません。

POLLING_TARGET

IP_MONITOR が ON に設定されている場合、先行する *SUBNET* エントリーで指定されたサブネット上のすべてのネットワークインターフェイスからのポーリングメッセージの送信先 IP アドレスです。これを、ターゲットポーリングといいます。

サブネットごとに複数のポーリングターゲットを設定できます。必要な数だけ *POLLING_TARGET* エントリーを繰り返します。

IP_MONITOR は ON に設定されているのに *POLLING_TARGET* が指定されていない場合、ポーリングメッセージは同じサブネット上のネットワークインターフェイス間で送信されます (ピアポーリング)。ターゲットポーリングを使用することをお勧めします。詳細は、「[IP モニターの動作](#)」(60 ページ) を参照してください。

注記: *cmquerycl* (1m) は、各ノードの経路指定テーブルでゲートウェイを探すことによってクラスター内の第 1 レベルのルーターを検出し、それをここにポーリングターゲットとしてリストします。*-w full* オプション (全ネットワークのプロロービング) を指定して *cmquerycl* を実行すると、ゲートウェイが監視用に正しく機能するかどうかを検証されます。

このパラメーターはクラスターの稼働中に変更することもできます。先行する *SUBNET* エントリーが削除された場合は削除しなければなりません。

`WEIGHT_NAME`, `WEIGHT_DEFAULT`

重みを指定できるすべてのパッケージにおける、この重みのデフォルト値です。「規則とガイドライン」(113 ページ)の「パッケージの重みについて」(108 ページ)を参照してください。`WEIGHT_NAME`は、クラスター構成ファイルで先に指定されている `CAPACITY_NAME` に**正確に一致する**重みの名前を指定します。(パッケージには重み、ノードにはキャパシティを指定します)。`WEIGHT_NAME`の形成規則は、このリストで前述の `CAPACITY_NAME` で説明した規則と同じです。

これらのパラメーターはオプションですが、定義する場合は、`WEIGHT_DEFAULT` を `WEIGHT_NAME` の後に指定し、0~1000000 までの浮動小数点値を設定する必要があります。特定の重みに対してこの値が指定されていない場合、Serviceguard はその重みをデフォルト値の 0 とみなします。どちらの場合も、`weight_name` と `weight_value` パラメーターをパッケージ構成ファイルに指定することによって、個々のパッケージでデフォルト値をオーバーライドすることができます。

詳細とサンプルは、「[重みの定義](#)」(111 ページ)を参照してください。

- ❶ **重要:** `CAPACITY_NAME`、`WEIGHT_NAME`、`weight_value` はすべて正確に一致する必要があります。

注記: 対応するキャパシティ (`CAPACITY_NAME`、`CAPACITY_VALUE`) がノードで定義されていない限り、重み (`WEIGHT_NAME`、`WEIGHT_DEFAULT`) はノードで無効です。

予約済みの重みとキャパシティ `package_limit` では、デフォルトの重みは常に 1 です。このデフォルト値はクラスター構成ファイルでは変更できませんが、パッケージ構成ファイルで個々のパッケージに対してオーバーライドできます。

重みにデフォルト値を定義したのにクラスター内の少なくとも 1 つのノードで同じ名前のキャパシティを指定しなかった場合、`cmapplyconf` は失敗します。1 つのクラスターには最大 4 個の `WEIGHT_DEFAULT` を定義できます。

このパラメーターはクラスターの稼働中に変更することもできます。

(アクセス制御ポリシー)

各ポリシーで、`USER_NAME`、`USER_HOST`、`USER_ROLE` の 3 つを指定します。クラスター構成ファイルとパッケージ構成ファイルで設定するポリシーには、競合や重複があってはなりません。詳細は、「[クラスターへのアクセスの制御](#)」(146 ページ)を参照してください。

`MAX_CONFIGURED_PACKAGES`

クラスター内に構成できるパッケージの最大数を設定します。最小値は 0、最大値とデフォルト値は 300 です。

このパラメーターはクラスターの稼働中に変更することもできます。

クラスター構成: 次の手順

クラスターを構成する準備ができたなら、「[クラスターの構成](#)」(141 ページ)に進みます。構成をあらかじめ記録したい場合は、クラスター構成用ワークシート(266 ページ)を使います。

パッケージ構成のプランニング

パッケージのプランニングには、各グループの高可用性サービスに関する情報の収集作業が含まれます。

注記: Serviceguard A.11.18 から、新しい簡単なパッケージ構成方法が提供されます。この方法を使うと、小規模なモジュールを組み合わせてパッケージを構築することができ、独立したパッケージ制御スクリプトが不要になり、そのスクリプトを手作業で配布する必要がなくなります。手順の詳細は、第6章「[パッケージとサービスの構成](#)」(157 ページ)を参照してください。

本書では、新しい方法で作成されたパッケージをモジュラーパッケージ、古い方法で作成されたパッケージを従来のパッケージと呼びます。

以降の説明は、モジュラー方式を使っていることを前提としています。従来のパッケージの作成と維持についての情報と手順は、「[従来のパッケージの構成](#)」(218 ページ)を参照してください。

『Framework for HP Serviceguard Toolkits』というドキュメントには、アプリケーションと Serviceguard の統合の手引きが示されています。従来のパッケージで使用するための、カスタマイズ可能なスクリプトのスイートも記載されています。このドキュメントは、<http://www.hp.com/go/softwaredepot> から無償でダウンロードできる Serviceguard Developer's Toolbox に含まれています。

注記: 本書の執筆時点では、『Framework for HP Serviceguard Toolkits』は従来のパッケージ専用です。

論理ボリュームとファイルシステムのプランニング

クラスター上でパッケージが動作するためのストレージインフラストラクチャとして、ボリュームグループ内で論理ボリュームを使います。一方のノードからもう一方のノードへパッケージが移動するとき、移動前のノードでアクセスしていたのと同じディスク上の同じデータにアクセスできなければなりません。このためには、ボリュームグループをアクティブ化してファイルシステムをマウントします。

Serviceguard では、高可用性アプリケーション、サービス、データは、共有バス上のボリュームグループに置かれます。ノードで障害が発生した場合、そのノードのアプリケーション、サービス、データを持つボリュームグループは、障害の発生したノード上で非アクティブ化され、引き継ぎノード(パッケージの移動先のノード)上でアクティブ化されます。このような動作をさせるためには、障害の発生したノードから引き継ぎノードへボリュームグループを移動できるように、ボリュームグループを構成しなければなりません。

注記: オペレーターがクラスター内の他のノード上でボリュームグループを誤ってアクティブ化しないように、バージョン A.11.16.07 以降の Serviceguard for Linux は、ある種の VG アクティブ化保護機能を備えています。これは、LVM2 の「ホストタグ」機能を基にしています。この機能は必須ではありませんが、既存のクラスターをアップグレードするときや、新しいクラスターを作成するときには、この機能を実装することを強くお勧めします。手順については、「[ボリュームグループのアクティブ化保護を有効にする](#)」(135 ページ)を参照してください。

プランニングの一部として、以下の項目を決定する必要があります。

- どのようなボリュームグループが必要か。
- どれくらいのディスクスペースが必要か。また、このスペースを論理ボリューム内でどのように割り当てるか。
- 各パッケージにどのファイルシステムのマウントが必要か。
- どのノードがどの論理ボリューム構成をインポートするか。
- パッケージが引き継ぎノードへ移動する場合、移動したパッケージは性能にどのように作用するか。

- パッケージの一部として監視する必要があるハードウェア/ソフトウェアリソースの特定。以上が決定したら、それらの情報をパッケージ内の汎用リソースとして構成して、適切な監視スクリプトを作成し、リソースを監視できます。

注記: 汎用リソースは、ステータスに基づきパッケージに影響を及ぼします。リソースの実際の監視は、スクリプトで行う必要があります。スクリプトはサービスとして構成する必要があります。スクリプトは、リソースを使用できるかどうかを判断してリソースのステータスを設定します。付録 G 「汎用リソース用の監視スクリプト」を参照してください。

ボリュームグループ、論理ボリュームおよびファイルシステムのリストをパッケージごとに作成してください。同じファイルシステムに対して別のタイミングでアクセスする必要のあるノードを明示してください。

論理ボリューム名をカスタマイズして、デフォルトの論理ボリューム名 (lv011、lv012 など) 以外の名前を使用することをお勧めします。関連する高可用性アプリケーションを示す名前 (lvoldatabase など) を付けておくと、クラスターが管理しやすくなります。

各ノード上の、パッケージに関連するボリュームグループ、論理ボリューム、ファイルシステムに関する詳しい説明を記述したいときには、/etc/fstab ファイルにコメント行を追加します。データベースアプリケーションの例を以下に示します。

```
# /dev/vg01/lvoldb1 /applic1 ext3 defaults 0 1 # These six entries are
# /dev/vg01/lvoldb2 /applic2 ext3 defaults 0 1 # for information purposes
# /dev/vg01/lvoldb3 raw_tables ignore ignore 0 0 # only. They record the
# /dev/vg01/lvoldb4 /general ext3 defaults 0 2 # logical volumes that
# /dev/vg01/lvoldb5 raw_free ignore ignore 0 0 # exist for Serviceguard's
# /dev/vg01/lvoldb6 raw_free ignore ignore 0 0 # HA package. Do not uncomment.
```

各ボリュームグループにエントリーを作成して、ファイルシステムまたは raw デバイス向けであることを明記してください。

△ 注意: Serviceguard パッケージが使用するファイルシステムのマウントには /etc/fstab ファイルを使用しないでください。

ボリュームグループの作成、エクスポート、インポートについては、「[論理ボリュームインフラストラクチャの作成](#)」(132 ページ)を参照してください。

NFS マウントされたファイルシステムのプランニング

Serviceguard A.11.20.00 では、パッケージの共有ストレージとして NFS マウント (インポート) ファイルシステムを使用できるようになりました。

同じパッケージが複数の NFS インポートファイルシステムをマウントでき、クラスターのローカル共有ストレージと NFS インポートの両方を使用できます。

以下の規則と制限があります。

- NFS マウントがサポートされるのは、モジュラー形式のフェイルオーバーパッケージです。
- Serviceguard では、パッケージで障害が発生したノードのすべての I/O がフラッシュされてから、パッケージが引き継ぎノードで再起動するため、NFS サーバーとクライアント間のすべてのネットワークスイッチとルーターは、最悪の状況でのタイムアウト (この時間を超えるとパケットとフレームが削除される) に対応する必要があります。このタイムアウトは、Maximum Bridge Transit Delay (MBTD) と呼ばれます。

- ① **重要:** 影響を受けるルーターとスイッチそれぞれの MBTD 値をベンダーのドキュメントで確認します。可能性のあるすべてのパスを判別し、それらのパスに基づき MBTD 値の最悪の場合の合計を求めます。その結果の値を使用して、Serviceguard の `CONFIGURED_IO_TIMEOUT_EXTENSION` パラメーターを設定します。手順は、「[クラスター構成のパラメーター](#)」(82 ページ) のこのパラメーターの説明を参照してください。MBTD 値をサポートしないスイッチとルーターは、Serviceguard の NFS 構成では使わないでください。使用した場合、パケットの遅延につながり、データ破損が引き起こされる恐れがあります。

- Serviceguard のノード間のネットワークは、ネットワークでの 1 回の障害がパッケージの障害を引き起こさないように構成する必要があります。
- NFS のクライアント側ロック (ローカルロック) のみがサポートされます。サーバー側のロックはサポートされません。
- NFS インポートファイルシステムでは排他的なアクティブ化を行えないため、データを誤って上書きしないように以下の予防策を行う必要があります。
 - クラスターのノードのみがファイルシステムにアクセスできるようにサーバーを構成する。
 - パッケージで使用される NFS ファイルシステムを、他のシステム (クラスター内の他のノードを含む) がインポートしないようにする。
 - ブート時に、ノードがファイルシステムをマウントしないようにする。マウントしてよいのは、ファイルシステムを使用するパッケージの起動時のみです。
 - NFS ファイルシステムは、1 つのパッケージのみで使用する。
 - パッケージが稼働している間、パッケージがファイルシステムを排他的に使用する。
 - パッケージで障害が発生した場合、ファイルシステムが適切にマウント解除されたことを確認できるまでは、手動でパッケージを再起動しない。

さらに、以下のガイドラインにも注意してください。

- CacheFS および AutoFS は、NFS マウントを使うパッケージを実行するように構成されたすべてのノードで無効にする必要があります。

HP-UX での NFS についての詳細は、<http://www.hp.com/go/hpux-networking-docs> にある『NFS Services Administrator's Guide HP-UX 11i version 3』を参照してください。
- NFS サーバーを高可用性にすることで、単一障害点を回避することをお勧めします。

注記: NFS サーバーへのネットワーク接続が切断されると、インポートされたファイルシステムを使用するアプリケーションがハングし、抹消できなくなる可能性があります。パッケージがこの時点で停止を試行しても、正常に停止できない場合があります。

- 自動マOUNTERを使わないでください。使うと、パッケージの起動が失敗することがあります。
- ストレージが、すべてのクラスターノードに直接接続されて共有されている場合は、NFS を使う代わりにストレージをローカルファイルシステムとして構成してください。
- NFS ファイルシステムは、複数のマウントポイントに同時にマウントしないでください。
- パッケージで使われる NFS ファイルシステムへのアクセスは、そのパッケージを実行できるノードに制限する必要があります。

詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> または <http://www.hp.com/go/linux-serviceguard-docs> にあるホワイトペーパー『Using NFS as a file system type with

Serviceguard 11.20 on HP-UX and Linux』を参照してください。このホワイトペーパーには、NFS インポートファイルシステムを使うサンプルパッケージの設定手順が含まれています。

また、[fs_name\(176 ページ\)](#)、[fs_type\(176 ページ\)](#)、およびその他のファイルシステムに関連するパッケージパラメーターの説明も参照してください。

拡張のプランニング

実行中のクラスターにパッケージを追加することができます。この手順については、[第7章「クラスターとパッケージの管理」\(186 ページ\)](#)で説明しています。

パッケージを追加するには、クラスター構成ファイルで定義されている `max_configured_packages` の値を上回らないように注意してください ([「クラスター構成のパラメーター」\(82 ページ\)](#)を参照)。必要であれば、クラスターの稼働中にこのパラメーターを変更することができます。

切り替え動作とフェイルオーバー動作の選択

フェイルオーバーパッケージ ([「パッケージのタイプ」\(38 ページ\)](#)を参照) のフェイルオーバー動作を設定するために、動作していないパッケージを Serviceguard が自動的に起動する場所を決定するポリシーを定義します。さらに、プライマリノードが使用可能になったときにパッケージをプライマリノードに自動的に戻すかどうかを指定するフェイルバックポリシーを定義します。

次の表では、各種のフェイルオーバー動作と、その動作を決定するパッケージ構成ファイルの設定について説明します。詳細は、[「パッケージのパラメーターの説明」\(162 ページ\)](#)を参照してください。

表 6 パッケージのフェイルオーバー動作

切り替え動作	構成ファイルのパラメーター
通常は、サービス、ネットワーク、汎用リソースの障害の検出後や、構成されている依存関係が満たされなかったときに、パッケージの切り替えが行われる。切り替えが行われる前に、停止スクリプトが実行される。(デフォルト)	<code>node_fail_fast_enabled</code> に <code>no</code> を設定。(デフォルト) - すべてのサービスの <code>service_fail_fast_enabled</code> を <code>no</code> に設定 (デフォルト) - パッケージに対し <code>auto_run</code> に <code>yes</code> を設定。(デフォルト)
アクティブなパッケージが最も少ないノードにパッケージがフェイルオーバーする。	<code>failover_policy</code> に <code>min_package_node</code> を設定。
ノードリスト上で次にあるノードにパッケージがフェイルオーバーする。(デフォルト)	<code>failover_policy</code> に <code>configured_node</code> を設定。(デフォルト)
プライマリノードが利用可能で、パッケージが非プライマリノードで実行されている場合に、パッケージが自動的に停止され、プライマリノード上で再起動される。	<code>failback_policy</code> に <code>automatic</code> を設定。
パッケージが一次ノード以外のノードで実行されている場合、パッケージは手動で一次ノードに戻ることができるが、自動では戻されない。	<code>failback_policy</code> に <code>manual</code> を設定。(デフォルト) <code>failover_policy</code> に <code>configured_node</code> を設定。(デフォルト)
特定のサービスで障害が発生した場合、ノード上のすべてのパッケージをシステムリブートに続いて切り替える。停止スクリプトは実行されない。	- 特定のサービスに対し <code>service_fail_fast_enabled</code> に <code>yes</code> を設定。 - すべてのパッケージに対し <code>auto_run</code> に <code>yes</code> を設定。
サービスで障害が発生した場合、ノード上のすべてのパッケージをシステムリブートに続いて切り替える。	- すべてのサービスに対し <code>service_fail_fast_enabled</code> に <code>yes</code> を設定。 - すべてのパッケージに対し <code>auto_run</code> に <code>yes</code> を設定。
特定のサービスで障害が発生したときに、ノード上でのシステムリセット (正常な	特定のサービスに対し <code>service_fail_fast_enabled</code> に <code>yes</code> を設定。

表 6 パッケージのフェイルオーバー動作 (続き)

切り替え動作	構成ファイルのパラメーター
シャットダウン手順なしの即時停止)後にすべてのパッケージが切り替えられる。停止スクリプトは実行されない。	すべてのパッケージに対し <code>auto_run</code> に <code>yes</code> を設定。
任意のサービスで障害が発生したときに、ノード上でのシステムリセット後にすべてのパッケージが切り替えられる。システムリセットの前に、まずリブートが試みられる。	- すべてのサービスに対し <code>service_fail_fast_enabled</code> に yes を設定。 - すべてのパッケージに対し <code>auto_run</code> に <code>yes</code> を設定。

フェイルオーバーパッケージは、障害が発生した NIC から、同じノード上で同じ物理サブネット上のスタンバイ NIC へ IP アドレス切り替えを行うように構成することもできます。

汎用リソースを構成するパラメーター

Serviceguard は、汎用リソース構成のために次のパラメーターを提供します。パッケージが依存している各リソースについて、これらのパラメーターをパッケージ構成ファイルに構成します。

- `generic_resource_name`: パッケージ内の汎用リソースの識別に使用する論理名を定義します。
- `generic_resource_evaluation_type`: 汎用リソースのステータス評価のタイミングを定義します。この値は、`during_package_start` または `before_package_start` に設定できます。指定しない場合、デフォルトで `DPS` とみなされます。
 - `during_package_start` は、汎用リソースのステータスがパッケージの一連の起動プロセスの中で評価されることを意味します。
 - `before_package_start` は、パッケージの起動前にリソース監視を開始しなければならないことを意味します。すなわち、あるノードでパッケージを起動するには、そのノードで構成されるすべてのリソースの状態が `UP` でなければなりません。
- `generic_resource_up_criteria`: ある汎用リソースが `UP` の状態にあるかどうかを判定するための基準を定義します。また、このパラメーターにより、汎用リソースがシンプルリソースかそれとも拡張リソースかも決定されます。このパラメーターには、演算子と値が必要です。演算子として使用できるのは、`==`、`!=`、`>`、`<`、`>=`、および `<=` です。値は、1～2147483647 の正の整数でなければなりません。

シンプルリソースと拡張リソースの構成方法の例を、次に示します。

シンプル汎用リソース：

```
generic_resource_name          sfm_disk
generic_resource_evaluation_type before_package_start
```

拡張汎用リソース：

```
generic_resource_name          cpu_lan
generic_resource_evaluation_type during_package_start
generic_resource_up_criteria    <50
```

汎用リソースのパラメーターについての詳細は、「[パッケージのパラメーターの説明](#)」(162 ページ) を参照してください。

汎用リソースの構成

この項では、汎用リソースの構成方法を手順に沿って説明します。汎用リソースは、Serviceguard Manager から構成することもできます。Serviceguard Manager からの構成方法については、オンラインヘルプを参照してください。

- 汎用リソースモジュールを含むパッケージ構成ファイルを作成します。

```
cmmakepkg $SGCONF/pkg1/pkg1.conf
```

Package template is created.
This file must be edited before it can be used.

注記: 既存のパッケージに汎用リソースモジュールを追加する構成ファイルを生成するには、以下のように指定します (コマンド全体を 1 行で入力します)。

```
cmmakepkg -i $SGCONF/pkg1/pkg1.conf -m sg/generic_resource
```

2. パッケージ構成ファイルを編集して、汎用リソースパラメーターを指定します (ファイルの一部を示しています)。

```
service_name      cpu_monitor
service_cmd       $SGCONF/generic_resource_monitors/cpu_monitor.sh
service_halt_timeout      10

generic_resource_name      sfm_cpu
generic_resource_evaluation_type      during_package_start
```

注記: 汎用リソースは、監視スクリプトを使用するように構成する必要があります。この監視スクリプトは、リソースを監視し、`cmsetresource(1m)` を使用して状況に合わせて汎用リソースのステータスを設定するためのロジックを含みます。

これらのスクリプトは、エンドユーザーが各自の要件に合わせて作成しなければなりません。監視スクリプトは、リソースの監視をパッケージの一部として開始および停止する必要がある場合は、パッケージ内のサービスとして構成する必要があります。

これは、手順の説明で示したように `service_name` と `service_cmd` を設定し、`service_cmd` の値として監視スクリプトのフルパス名を入力することで実現できます。`service_name` と `generic_resource_name` を一致させる必要はありませんが、一致させるとモニターの識別が容易になるため、便利な方法です。

監視スクリプトの作成方法を説明するテンプレートが用意されています。監視スクリプトおよびテンプレートの詳細は、「[汎用リソース用の監視スクリプト](#)」(279 ページ) および「[監視スクリプトのテンプレート](#)」(281 ページ) を参照してください。

`generic_resource_up_criteria` を指定すると、そのリソースは拡張汎用リソースと判断され、指定しなければシンプル汎用リソースと判断されます。汎用リソースのパラメーターの説明は、「[パッケージのパラメーターの説明](#)」(162 ページ) に掲載しています。「[汎用リソース監視サービスの使用](#)」(45 ページ) を参照してください。

3. パッケージ構成ファイルの編集が完了したら、ファイルの内容を検証します。

```
cmcheckconf -v -P $SGCONF/pkg1/pkg1.conf
```

```
cmcheckconf: Verification completed with no errors found.
Use the cmapplyconf command to apply the configuration
```

4. 検証がエラーなしで完了した場合には、パッケージ構成ファイルを適用します。これによりパッケージ構成情報 (汎用リソースとともに) が `$SGCONF` ディレクトリ内のバイナリ形式クラスター構成ファイルに追加され、すべてのクラスターノードに配布されます。

```
cmapplyconf -P $SGCONF/pkg1/pkg1.conf
```

```
Modify the package configuration ([y]/n)? y
Completed the cluster update
```

5. 汎用リソースパラメーターが設定されていることを確認します。

```
cmviewcl -v -p pkg1
```

```
UNOWNED_PACKAGES
```

PACKAGE	STATUS	STATE	AUTO_RUN	NODE
pkg1	down	halted	disabled	unowned

```
Policy_Parameters:
```

POLICY_NAME	CONFIGURED_VALUE
Failover	configured_node
Failback	manual

Script_Parameters:

ITEM	STATUS	NODE_NAME	NAME
Generic Resource	unknown	node1	sfm_disk
Generic Resource	unknown	node2	sfm_disk

Node_Switching_Parameters:

NODE_TYPE	STATUS	SWITCHING	NAME
Primary	up	enabled	node1
Alternate	up	enabled	node2

Other_Attributes:

ATTRIBUTE_NAME	ATTRIBUTE_VALUE
Style	modular
Priority	no_priority

cmviewcl -v -f line は、次のような出力を生成します (一部を抜粋)。

```
cmviewcl -v -f line -p pkg1 | grep generic_resource
```

```
generic_resource:sfm_disk|name=sfm_disk
generic_resource:sfm_disk|evaluation_type=during_package_start
generic_resource:sfm_disk|up_criteria="N/A"
generic_resource:sfm_disk|node:node1|status=unknown
generic_resource:sfm_disk|node:node1|current_value=0
generic_resource:sfm_disk|node:node2|status=unknown
generic_resource:sfm_disk|node:node2|current_value=0
```

注記: cmsetresource コマンドでシンプル/拡張汎用リソースのステータスと値が設定されていない場合、汎用リソースのデフォルトステータスは UNKNOWN、デフォルト current_value は、「0」です。

6. パッケージを起動します。監視スクリプトは、パッケージの起動プロセスで汎用リソースの監視を始め、状況に合わせてステータスを設定します。

```
cmrunpkg pkg1
```

シンプル/拡張汎用リソースのステータスと値の取得と設定

シンプル汎用リソースのステータスや拡張汎用リソースの値の取得と設定には、Serviceguard コマンドの cmgetresource(1m) と cmsetresource(1m) を使用できます。これらのコマンドは、監視スクリプトで使用することも CLI から実行することもできます。これらのコマンドを実行するユーザーは、root ユーザー (UID=0) でなければなりません。root ユーザー以外のユーザーは、これらのコマンドを実行できません。

Serviceguard コマンドを使用したシンプル/拡張汎用リソースのステータスと値の取得

シンプル汎用リソースのステータスまたは拡張汎用リソースの値の取得には、cmgetresource コマンドを使用します。例:

```
cmgetresource -r sfm_disk
```

汎用リソース sfm_disk がシンプルリソースとして構成されている場合、このコマンドにより汎用リソースのステータスが取得されます。拡張リソースとして構成されている場合は、現在の値が返されます。

Serviceguard コマンドを使用したシンプル/拡張汎用リソースのステータスと値の設定

シンプル汎用リソースのステータスまたは拡張汎用リソースの値の設定には、cmsetresource コマンドを使用します。例:

```
cmsetresource -r sfm_disk -s up
```

このコマンドにより汎用リソース `sfm_disk` のステータスが「up」に設定されます。このリソースはシンプル汎用リソースであり、ステータスとして設定が許可されるのは「up」または「down」だけです。

```
cmsetresource -r sfm_lan 10
```

このコマンドにより、汎用リソース `sfm_lan` の現在の値が「10」に設定されます。このリソースは拡張リソースであり、設定が許可されるのは 1～2147483647 の数字だけです。

詳細は、マンページを参照してください。

汎用リソースのオンライン再構成

追加、削除、変更など、パッケージ内の汎用リソースのオンライン操作がサポートされています。以下の操作をオンラインで実行できます。

- `generic_resource_evaluation_type` が `during_package_start` に設定され、ステータスが Down でない汎用リソースの追加。
汎用リソースを追加する際に、同等のモニターを利用できることを確認してください。そうでない場合は、汎用リソースを追加する際にモニターを追加してください。
- `generic_resource_evaluation_type` が `before_package_start` に設定され、ステータスが「up」の汎用リソースの追加。
- 汎用リソースの削除。汎用リソースを削除する際に、同等のモニターも削除されることを確認してください。ただし、複数のパッケージで共通のリソースを監視している場合は、モニターを削除する前に、削除する汎用リソースが、このモニターを使用している他のパッケージで構成されていないことを確認してください。
- リソースが「up」の状態での `generic_resource_evaluation_type` の `before_package_start` と `during_package_start` の間での切り替え。
- `evaluation_type` が `before_package_start` または `during_package_start` のリソースで指定されている `generic_resource_up_criteria` の変更。ただし、新しい `up_criteria` によりリソースステータスが「down」と評価されない (すなわち、リソースの `current_value` が変更後も新しい `up_criteria` を満たす) ことが前提です。
- リソースタイプのシンプルリソースから拡張リソースへの変更。ただし、現在リソースを使用している稼働中のすべてのパッケージで `generic_resource_evaluation_type` が `during_package_start` でなければなりません。

パッケージの依存関係について

パッケージは他のパッケージへの依存関係 (つまり、被依存のパッケージがノード上で実行されない限り、パッケージは起動されない) を持つことができます。

第 6 章の「[dependency_condition](#)」(168 ページ) で説明している条件に従えば、同じクラスターノード上で実行されている他の任意のパッケージに対して、パッケージの依存関係を設定できます。

Serviceguard では 2 つの新機能が追加されました。依存対象となるパッケージの実行場所を大まかに指定する機能と、パッケージが停止中でなければならぬと指定する機能です。これらの機能については、この項の後の方の「[拡張依存関係](#)」(105 ページ) で説明します。先に、次の項 (「[単純依存関係](#)」(100 ページ)) を参照してください。

単純依存関係

単純依存関係は、あるパッケージが同じノード上での別のパッケージの実行を必要とする場合に発生します。このような条件を定義するには、`dependency_condition` と `dependency_location` の 2 つのパラメーターを使用し、それぞれリテラル値 `UP` と `same_node` を指定します。(構成の詳細については、「[dependency_name](#)」(167 ページ) 以降のパッケージパラメーターの定義を参照してください。拡張依存関係の詳細は、「[拡張依存関係](#)」(105 ページ) を参照してください。

あるパッケージが、他のパッケージが提供するサービスなしでは機能できない (または機能してはならない) 場合に、他のパッケージに対するパッケージの依存関係を設定してください。たとえば、pkg1 が、pkg2 が管理しているデータベースへのリアルタイムなインターフェイスを持つ Web システムだとします。この場合、pkg1 は pkg2 に依存しているといえます。パッケージ間の依存関係を設定するかどうかを検討する際には、以下の[単純依存関係の規則と単純依存関係のガイドライン \(104 ページ\)](#)を参考にしてください。

単純依存関係の規則

pkg1 を pkg2 に依存させたいとします。

注記: pkg1 は、他の複数のパッケージに依存することができます。pkg2 も、他の 1 つ以上のパッケージに依存することができます。ここでは規則をできるだけ分かりやすくするために、パッケージは 2 つだけであるとします。

- pkg1 は、pkg2 がノード上で実行中でなければ、どのノード上でも起動されません。
- pkg1 の `package_type`([163 ページ](#))と `failover_policy`([166 ページ](#))は、以下のよう
に、pkg2 のタイプと特性を制限します。
 - pkg1 がマルチノードパッケージの場合、pkg2 はマルチノードパッケージまたはシステムマルチノードパッケージでなければなりません。(システムマルチノードパッケージは一般的な用途にはサポートされていないことに注意してください。)
 - pkg1 がフェイルオーバーパッケージで、その `failover_policy` が `min_package_node` の場合、pkg2 はマルチノードパッケージまたはシステムマルチノードパッケージでなければなりません。
 - pkg1 がフェイルオーバーパッケージで、その `failover_policy` が `configured_node` の場合、pkg2 は次のいずれかでなければなりません。
 - マルチノードパッケージまたはシステムマルチノードパッケージ。**または**
 - `failover_policy` が `configured_node` のフェイルオーバーパッケージ。
- pkg2 は、`failover_policy` が `min_package_node` のフェイルオーバーパッケージであってはなりません。
- pkg2 の `node_name` リスト([164 ページ](#))には、pkg1 のすべてのノードが含まれていなければなりません。
 - つまり、pkg1 がクラスター内の任意のノードで稼働するように構成されている場合は (*), pkg2 も任意のノードで稼働するように構成する必要があります。

注記: pkg1 がアスタリスク (*) を使用するのではなく、すべてのノードをリストしている場合は、pkg2 もすべてのノードをリストする必要があります。

- `failover_policy` が `configured_node` のパッケージ間の依存関係の場合、できればノードを同じ順序でリストしてください。同じ順序でリストされていないと、`cmcheckconf` と `cmapplyconf` で警告が表示されます。
- パッケージは、直接的であっても間接的であっても、自分自身に依存することはできません。
つまり、pkg1 は自分自身を `dependency_condition`([168 ページ](#))に指定することはできません。また、pkg2 が pkg1 に依存している場合や、pkg2 が pkg3 に依存していてその pkg3 が pkg1 に依存しているといった場合は、pkg1 で pkg2 への依存を指定することはできません。
- pkg1 がフェイルオーバーパッケージであり、pkg2 がマルチノードパッケージまたはシステムマルチノードパッケージで、pkg2 に障害が発生した場合、pkg1 は停止し、その

`node_name` リスト上の次のノード (pkg2 が動作し、リソースの依存関係や第 3 のパッケージへの依存関係など、他の依存関係を満たしている) にフェイルオーバーします。

- `failover_policy` が `configured_node` のフェイルオーバーパッケージの場合、一連の規則により、特定のノード上での pkg2 の起動を pkg1 が強制できる状況を指定します。これをドラッグといい、各パッケージの `priority` (167 ページ) で決定されます。「単純依存関係のドラッグ規則」 (102 ページ) を参照してください。
- pkg2 に障害が発生すると、Serviceguard は pkg1 を停止し、また pkg2 に直接的または間接的に依存している他のパッケージを停止します。

Serviceguard はデフォルトでは、依存しているパッケージをまず停止してから、そのパッケージに依存されているパッケージを停止するというように、依存順にパッケージを停止します。この例では、まず pkg1 が停止してから、pkg2 が停止します。pkg1 に依存する第 3 のパッケージ pkg3 が存在する場合、pkg3 がまず停止してから、pkg1 が停止し、次に pkg2 が停止します。

依存しているパッケージの停止スクリプトがハングアップすると、依存されているパッケージは、デフォルトでは永久に待ち状態になります (pkg2 は pkg1 を永久に待ち、pkg1 に依存する pkg3 が存在する場合、pkg1 は永久に pkg3 を待ちます)。この動作は `successor_halt_timeout` パラメーター (166 ページ) を使って変更することができます。(パッケージの後続パッケージは、そのパッケージに依存しています。この例では、pkg1 は pkg2 の後続パッケージです。逆に pkg2 は、pkg1 の先行パッケージと言えます。)

単純依存関係のドラッグ規則

`priority` パラメーター (167 ページ) を使うと、`failover_policy` が `configured_node` のフェイルオーバーパッケージのセットにおいて、1 つ以上のパッケージが他のパッケージに依存しているときに、起動、フェイルオーバー、およびフェイルバックの動作に影響を与えることができます。

優先順位の高いパッケージが、優先順位の低いパッケージをドラッグし、優先順位の高いパッケージに合ったノードで起動させたり、そのノードへ移動できるというのが、大まかな規則です。

注記: この項目は、パッケージが自動的に起動された (パッケージ切り替えが使用可能である) 場合にのみ適用されます。cmrunpkg で起動したパッケージが強制的に停止されることはありません。

他のパッケージに依存するパッケージが存在する場合でも、`priority` を設定する**必要**はありません。多くの場合、デフォルト値の `no_priority` で、ユーザーが期待する動作となります。たとえば、pkg1 が pkg2 に依存しており、どちらのパッケージの `priority` にも `no_priority` が設定されていて、この項で推奨したとおりに `node_name` や `auto_run` などの他のパラメーターが設定されている場合、両方のパッケージを実行できるときには、pkg1 は通常、pkg2 の後に実行されます。これは、常識的な (最も望ましい) 動作です。

以下の例は、`failover_policy` (166 ページ) が `configured_node` に設定された 2 つのフェイルオーバーパッケージに適用される規則について示しています。pkg1 が pkg2 に依存していて、`node1`、`node2`、および `node3` が各パッケージの構成ファイルの `node_name` (164 ページ) に一定の順序ですべて指定されており、また各パッケージの `failback_policy` (167 ページ) が `automatic` に設定されているものとします。

注記: 下記の例を参照する際と、実際に優先順位を構成する際には、以下の事項に注意してください。

1. 関連するすべてのパッケージの `auto_run` (164 ページ) には、`yes` が設定されていなければなりません。この例では、この設定を前提としています。
2. 優先順位は、ランキングの順序を示しています。そのため、小さい値ほど優先順位は高くなります (たとえば、10 は 30 よりも優先順位が高い)。
シーケンスに間隔を空けるために、値を 20 きざみで割り当てることをお勧めします。そうしないと、新しいパッケージに優先順位を割り当てるときに、既存の優先順位をすべて変更しなければならなくなることがあります。
デフォルトの `no_priority` は、どの数値よりも低い優先順位として扱われます。
3. `no_priority` のパッケージはすべて同じ優先順位です。同じ優先順位を割り当てる方法は、他にはありません。数値の優先順位は、同じクラスター内で重複があってはなりません。詳細は、「`priority`」 (167 ページ) を参照してください。

`pkg1` が `pkg2` に依存し、`pkg1` の優先順位が `pkg2` の優先順位以下で、`pkg2` のノード順序が優先される場合、`pkg2` のノード順序は、`node1`、`node2`、`node3` とします。

- 起動時:
 - `pkg2` は、`node1` で起動されます。`node1` が利用できないか、依存関係のすべての条件を現在満たしていないなどの場合は、`node2` で起動されます。
 - `pkg1` の他の依存関係をすべて満たしていれば、`pkg1` は `pkg2` が起動されたノードで (`pkg1` の `node_name` リストのどこにそのノードが指定されているかにかかわらず) 起動されます。
 - `pkg2` が起動されたノードが、`pkg1` の依存関係のすべてを満たしていない場合は、`pkg1` は起動されません。
- フェイルオーバー時:
 - `node1` 上で `pkg2` に障害が発生すると、`pkg2` は `node2` に (または、`node2` が利用できないか、依存関係のすべての条件を現在満たしていないなどの場合は、`node3` に) フェイルオーバーします。
 - `pkg1` の依存関係をすべて満たしていれば、`pkg1` は `pkg2` が再起動されたノードへ (`pkg1` の `node_name` リストのどこにそのノードが指定されているかにかかわらず) フェイルオーバーします。
 - `pkg2` が再起動されたノードが、`pkg1` の依存関係のすべてを満たしていない場合は、`pkg1` は再起動されません。
 - `pkg1` で障害が発生した場合、`pkg1` はフェイルオーバーしません。
これは、`pkg2` がいずれかの引き継ぎノード上で実行されるまでは `pkg1` をそのノード上で再起動できませんが、`pkg2` が元のノード上で実行されたままとなっているためです。`pkg1` は、優先順位が十分でないため、`pkg2` をドラッグすることはできません。
- フェイルバック時:
 - 両方のパッケージが `node1` から `node2` に移動され、`node1` が使用可能になると、`pkg2` は、**`pkg2` の優先順位が `pkg1` の優先順位より高い場合にだけ**、`node1` にフェイルバックします。
 - 優先順位が等しい場合、どちらのパッケージもフェイルバックしません (ただし、`pkg1` が実行されていない場合、`pkg2` はフェイルバックできます)。

- pkg2 の優先順位が pkg1 の優先順位よりも高い場合、pkg2 は node1 にフェイルバックし、pkg1 の他の依存関係がすべてそのノードで満たされれば、pkg1 は node1 にフェイルバックします。
- pkg2 が node1 にフェイルバックしたが、node1 が pkg1 の依存関係をすべて満たさなければ、pkg1 は停止します。

pkg1 が pkg2 に依存し、pkg1 の優先順位が pkg2 の優先順位より高く、pkg1 のノード順序が優先される場合。pkg1 のノード順序は、node1、node2、node3 とします。

- 起動時:

- pkg1 は、起動場所として node1 を選択します。
- pkg2 は、node1 で動作可能であれば、node1 で (pkg2 の `node_name` リストのどこに node1 が指定されているかにかかわらず) 起動されます。
- pkg2 は、他のノード上ですでに実行されている場合、node1 で動作可能であれば、node1 にドラッグされます。
- pkg2 を node1 上で起動できなければ、パッケージは両方とも node2 (このノードでも起動できなければ、次のノード) で起動されます。

ノードは、pkg1 の `node_name` リスト順で試されます。pkg2 は、現在他のノードで実行されているかどうかにかかわらず、このリスト上の、最初に適したノードにドラッグされます。

- フェイルオーバー時:

- node1 上で pkg1 に障害が発生すると、pkg1 は node2 (または、node3 で動作可能で、node2 が利用できないか、依存関係のすべての条件を現在満たしていないなどの場合は、node3) をフェイルオーバー用に選択します。
- pkg2 は、pkg1 が選択したノードにドラッグされ、そこで再起動されます。その後、pkg1 がそのノードで再起動されます。

- フェイルバック時:

- 両方のパッケージが node2 に移動され、node1 が利用可能になったときは、そのノードで両方のパッケージが動作可能であれば、pkg1 は node1 にフェイルバックします。
- これ以外の場合は、どちらのパッケージもフェイルバックしません。

単純依存関係のガイドライン

単純依存関係のドラッグ規則の説明から、pkg1 が pkg2 に依存しているときに、pkg1 に高い優先順位を割り当てることは、場合によっては良い方法です。これにより、pkg1 で障害が発生したときのフェイルオーバー (およびフェイルバック) が成功する可能性が最も高くなります。

ただし、パッケージの相対的な重要度も考慮する必要があります。ビジネスのかなめとなるデータベースを pkg2 が実行している場合は、依存しているアプリケーションパッケージに何が起こっても、データベースの実行に支障がない方がよいと考えられます。このような場合は、データベースパッケージを最高の優先順位とします。

優先順位が設定されていない場合、ドラッグ規則では、他パッケージから依存されているパッケージを、そのパッケージに依存しているパッケージよりも優先します。

依存するパッケージと依存されるパッケージの重要度が現実的にほぼ同じ場合は、依存する側のパッケージに高い優先順位を割り当ててください。同じでない場合は、重要なパッケージに高い優先順位を割り当てるか、両方のパッケージの優先順位をデフォルトのままとしてください。

また、パッケージの障害時に何が発生するかも考えなければなりません。他のパッケージがそのパッケージに依存している場合、Serviceguard はこれらのパッケージ (および、これらのパッケージに依存しているパッケージ) を停止します。この状況は、障害が発生したパッケージの優先順位にかかわらず発生します。

デフォルトでは、パッケージは起動されたときと逆の順序で停止されます。依存する側のパッケージの停止スクリプトがハングアップした場合、障害が発生したパッケージは自身の停止処理の完了を無限に待ちます。これにより、依存するすべてのパッケージを完全に停止させることができるでしょうが、ユーザーが望んでいる動作ではないでしょう。この動作は、`successor_halt_timeout` パラメーター(166 ページ) を使って変更することができます。

`successor_halt_timeout` にゼロを設定すると、Serviceguard は依存する側のパッケージを、障害が発生したパッケージと同時に停止します。このパラメーターに正の数を設定すると、Serviceguard は起動時と逆の順番でパッケージを停止しますが、`successor_halt_timeout` 秒後には、依存する側のパッケージが停止スクリプトを完了したかどうかにかかわらず、障害が発生したパッケージを停止させることができます。

パッケージ間の依存関係を作成する場合は、パッケージを実運用環境で動作させる前に、完全にテストを行って、パッケージの起動、停止、フェイルオーバー、およびフェイルバック動作が期待したとおりになっていることを確認することをお勧めします。

拡張依存関係

単純依存関係 (100 ページ) の提供する機能に加えて、拡張依存関係は、以下の機能を提供します。

- 依存されているパッケージが実行中でなければならないか、または停止中でなければならないかを指定できます。

この条件を定義するには、`dependency_condition` を使って、リテラルUP またはDOWN (リテラルは大文字でも小文字でも可) のいずれかを指定します。別のパッケージが停止中でなければならないという要件を排他的依存関係と呼びます。「[排他的依存関係の規則](#)」(106 ページ) を参照してください。

- `dependency_condition` を満たす必要のある場所を、同じノード上、別のノード上、すべてのノード上、クラスター内の任意のノード上のいずれかに指定できます。

この条件を定義するには、`dependency_location` パラメーター(168 ページ) を使って、リテラル `same_node`、`different_node`、`all_nodes`、`any_node` のいずれかを指定します。

`different_node` と `any_node` は `dependency_condition` がUP のときだけ許可され、`all_nodes` は `dependency_condition` がDOWN の場合のみ許可されます。

「[different_node と any_node の依存関係の規則](#)」(106 ページ) を参照してください。

`dependency_` のパラメーターの詳細は、「[dependency_name](#)」(167 ページ) 以降の定義と `cmmakepkg (1m)` のマンページを参照してください。

-
- ❗ **重要:** 単純依存関係 (100 ページ) の説明にまだ目を通していない場合は、次に進む前にお読みください。
-

`dependency_location` と `dependency_condition` の有効な値の組み合わせによって、以下のことが可能になります。

- 同一ノードの依存関係: パッケージは別のパッケージが同じノード上でUPであることを要求できます。
このケースについては、[単純依存関係 \(100 ページ\)](#) の項で説明しています。
- 異なるノードの依存関係: パッケージは、別のパッケージが異なるノード上でUPであることを要求できます。
- 任意のノードの依存関係: パッケージは別のパッケージがクラスター内の任意のノード上でUPであることを要求できます。

- 同一ノードの排他的依存関係: パッケージは別のパッケージが同じノード上でDOWNであることを要求できます。(ただし、そのパッケージが他のノードでUPであってもかまいません。)
- すべてのノードの排他的依存関係: パッケージは別のパッケージがクラスター内のすべてのノード上でDOWNであることを要求できます。

排他的依存関係の規則

- 排他関係はすべて相互排除である必要があります。
つまり、pkg1 にとって pkg2 がDOWN である必要がある場合、pkg2 にとっても pkg1 がDOWN である必要があります。
2 つのパッケージ間に排他的依存関係を確立することによって、特定のノード (同一ノードの排他的依存関係) またはクラスター全体 (すべてのノードの排他的依存関係) を通してどちらか 1 つのパッケージだけが実行されることを保証できます。1 つのパッケージには、任意の数の別のパッケージとの排他関係を設定できますが、関係は必ず相互排除になります。
- 排他的依存関係では、少なくとも 1 つのパッケージの優先順位 (詳細は「[単純依存関係のドラッグ規則](#)」(102 ページ) で後述) を設定する必要があります。
優先順位の高いパッケージは優先順位の低いパッケージを強制的に停止させたり、(同一ノードの排他的関係の場合) 別の使用可能なノードがあればそのノードに移動させることができます。
- `dependency_location` は `same_node` または `all_nodes` のどちらかでなければならず、両方のパッケージで同じでなければなりません。
- 両方のパッケージとも、`failover_policy` (166 ページ) が `configured_node` であるフェイルオーバーパッケージでなければなりません。

different_node と any_node の依存関係の規則

以下の規則は、`dependency_condition` がUP で、`dependency_location` が `different_node` または `any_node` のパッケージに適用されます。同一ノードの依存関係については、[単純依存関係](#) (100 ページ) を参照してください。排他的依存関係については、「[排他的依存関係の規則](#)」(106 ページ) を参照してください。

- 両方のパッケージとも、`failover_policy` (166 ページ) が `configured_node` であるフェイルオーバーパッケージでなければなりません。
- 依存されているパッケージの `priority` (167 ページ) は、依存しているパッケージ、さらにそのパッケージに依存しているパッケージの優先順位と同じかそれより高くなければなりません。
 - たとえば、pkg1 がpkg2 に対して `different_node` または `any_node` の依存関係を持つ場合、pkg2 の優先順位は pkg1 および pkg1 に依存にしてUP 状態になるすべてのパッケージの優先順位と同じかそれより高くなければなりません。Serviceguard がパッケージを配置する際、pkg2 のノード順序が優先されます。
- パッケージは、直接的であっても間接的であっても、自分自身に依存することはできません。
たとえば、pkg1 は自分自身を `dependency_condition` (168 ページ) に指定してはならないだけでなく、pkg2 が pkg1 に依存している場合や、pkg2 が pkg3 に依存していて pkg3 が pkg1 に依存している場合も、pkg1 では pkg2 への依存関係を設定してはなりません。
- 「ドラッグ」規則が適用されます。「[単純依存関係のドラッグ規則](#)」(102 ページ) を参照してください。

パッケージに障害が発生したときの動作

この説明は、依存するパッケージがあるパッケージ、または依存されている側のパッケージ、または両方 (UP 依存関係のみ) に適用されます。このようなパッケージに障害が発生した場合、Serviceguard は以下のことを行います。

1. 障害の発生したパッケージに依存するパッケージがあれば、停止します。

Serviceguard は依存する側のパッケージ (および、これらのパッケージに依存するパッケージ) を停止します。この状況は、障害が発生したパッケージの優先順位にかかわらず発生します。

注記: 依存しているパッケージは、依存関係が `different_node` または `any_node` の場合でも停止されます。たとえば、`node1` で動作中の `pkg1` が `node2` で動作中の `pkg2` に対して `different_node` または `any_node` 依存関係があり、`pkg2` が `node3` にフェイルオーバーした場合、`pkg1` は停止され、以下のように再起動されます。

デフォルトでは、パッケージは起動されたときと逆の順序で停止されます。依存する側のパッケージの停止スクリプトがハングアップした場合、障害が発生したパッケージは自身の停止処理の完了を無限に待ちます。これにより、依存するすべてのパッケージを完全に停止させることができるでしょうが、ユーザーが望んでいる動作ではないでしょう。この動作は、`successor_halt_timeout` パラメーター (166 ページ) を使って変更することができます。(successor とは、後続という意味で、別のパッケージに依存するパッケージです。)

障害が発生したパッケージの `successor_halt_timeout` にゼロを設定すると、Serviceguard は依存する側のパッケージを、障害が発生したパッケージと同時に停止します。このパラメーターに正の数を設定すると、Serviceguard は起動時と逆の順番でパッケージを停止しますが、`successor_halt_timeout` 秒後には、依存する側のパッケージが停止スクリプトを完了したかどうかにかかわらず、障害が発生したパッケージを停止させることができます。

2. 障害が発生しているパッケージを停止します。

後続パッケージ停止タイマーの期限が終了するか、もしくは、依存する側のパッケージがすべて停止されると、Serviceguard は、依存する側のパッケージの停止が成功、失敗、またはタイムアウトのいずれの結果であっても、障害の発生したパッケージの停止スクリプトの実行を開始します。

3. 障害の発生したパッケージが直接依存するパッケージから順に、このパッケージが依存するパッケージを停止します。パッケージは以下の場合にのみ停止されます。

- フェイルオーバーパッケージであり、**かつ**
- 障害の発生したパッケージが、対象のパッケージがすべて実行できるノード上にパッケージを「ドラッグ」できる場合。

それ以外の場合、障害の発生したパッケージは停止し、依存しているパッケージは引き続き実行されます。

4. 障害の発生したパッケージが依存するパッケージがあれば (ステップ 3 で停止したもの)、起動します。

障害の発生したパッケージが、自らが依存するパッケージを引き継ぎノードにドラッグすることが可能であった場合、Serviceguard は前の手順で停止したのと逆の順番でそれらのパッケージを起動します (つまり、他のパッケージにまったく依存しないパッケージから先に起動されます)。

5. 障害が発生しているパッケージを起動します。
6. 障害の発生したパッケージに依存するパッケージ (ステップ 1 で停止したもの) を起動します。

詳細情報

詳細は、以下を参照してください。

- [priority\(167 ページ\)](#) と [dependency_\(167 ページ\)](#) のパラメーターの説明、およびパッケージ構成テンプレートファイルの対応するコメント
- `cmmakepkg` (1m) のマンページ
- ホワイトペーパー『Serviceguard's Package Dependency Feature』。このホワイトペーパーは <http://www.hp.com/go/hpux-serviceguard-docs> で入手できます。

パッケージの重みについて

パッケージ重みとノードキャパシティによって、ノード上で同時に実行されるパッケージの数を制限したり、あるいは、ノードが耐えられるパッケージの「重み」合計 (リソースの消費量という点で) を制限することができます。

たとえば、大規模なシステムと小規模なシステムからなる 2 ノードクラスターがあると仮定します。すべてのパッケージを大規模なシステムで同時に実行できるようにしますが、大規模なノードに障害が発生した場合、重要なパッケージだけを小規模なシステムで実行する必要があります。パッケージ重みを使うことで、Serviceguard にこのような動作をさせることができます。

パッケージ重みとノードキャパシティ

ノードのキャパシティをクラスター構成ファイルで、対応するパッケージの重みをパッケージ構成ファイルで定義します。

ノードキャパシティはパッケージ重みによって消費されます。Serviceguard は、ノードで実行されるパッケージの合計の重みがノードに対して設定したキャパシティの制限を決して超えないようにします。ノード上での実行を希望するパッケージがそのノードで使用可能なキャパシティを超えた場合、パッケージはそのノードでは実行されません。したがって、たとえば、現在利用可能なキャパシティがないノードには、パッケージはフェイルオーバーすることができません。たとえ、他の場合にはパッケージを実行できるノードであっても、実行しようとするパッケージの優先順位が、現在動作中のパッケージのいずれかを強制的に移動させるだけの優先順位を持っていない限り、フェイルオーバーできません。「[パッケージ重みとパッケージの優先順位および依存関係との相互作用](#)」(114 ページ) を参照してください。

重みとキャパシティの構成

ノードに対して複数のキャパシティを構成でき、対応するパッケージに対しても複数の重みを構成できます (1 クラスターに最高 4 個のキャパシティと重みのペア)。したがって、パッケージによる各ノードのリソースの使用をかなり柔軟に管理できます。しかし、必要以上に柔軟すぎるかもしれません。このため、Serviceguard では、キャパシティと重みを構成するにあたり、単純な方法と包括的な方法の 2 種類の方法を提供しています。これらの方法について、以下に説明します。

単純な方法

単純に、ある時間に特定のノードで実行できるパッケージの数を制御したい場合には、この方法を使用してください。この方法は、すべてのパッケージがほぼ同量の演算リソースを消費する場合に最適に働きます。

リソースの消費の点でパッケージをより細かく区別する必要がある場合は、代わりに[包括的な方法](#) (110 ページ) を使用します。

単純な方法を実装するには、予約済みのキーワード `package_limit` を使用して各ノードキャパシティを定義します。この場合、Serviceguard では、この単一タイプのキャパシティと、対応するパッケージ重みのみをこのクラスターで定義することができます。パッケージ重みの定義はオプションです。パッケージ構成ファイルで変更しなければ、すべてのパッケージで、`package_limit` に対して、パッケージ重みはデフォルトの 1 となります。

例 1

たとえば、1 度に最大 10 個のパッケージを実行するようにノードを構成するには、クラスター構成ファイルのノードの `NODE_NAME` エントリーで以下のエントリーを作成します。

```
NODE_NAME node1
...
CAPACITY_NAME package_limit
CAPACITY_VALUE 10
```

これで、リソースの消費という点ではすべてのパッケージが等しいものとみなされ、このノードでは一度に 10 個より多くのパッケージは実行されなくなります。(次の例に示すように、必要があれば、一部またはすべてのパッケージの重みを変更することによってこの動作を変更できます。) 次に、残りのノードについて `CAPACITY_NAME` と `CAPACITY_VALUE` のパラメーターを定義します。それぞれの場合について、`CAPACITY_NAME` に `package_limit` を設定します。各ノードで、`CAPACITY_VALUE` を異なる値にする場合もあるでしょう。一例ですが、10 パッケージのキャパシティというのは非常に強力なノードであり、一方、性能の劣るノードでは、キャパシティはわずか 2、3 しかありません。

注記: Serviceguard では、各ノードについてキャパシティを定義する必要はありません。一部のノードに対して `CAPACITY_NAME` と `CAPACITY_VALUE` パラメーターを定義したが他のノードについては定義しなかった場合、パラメーターが定義されていないノードのキャパシティは無制限であるとみなされます。この場合、それらのノードでは、特定の時点で使用可能なパッケージをいくつでも実行できることになります。

一部のパッケージが他のパッケージよりも多くのリソースを消費する場合、`weight_name` と `weight_value` パラメーターを使って、一部またはすべてのパッケージのデフォルト値 (1) をオーバーライドすることができます。たとえば、`pkg1`、`pkg2`、`pkg3` の 3 つのパッケージがあって、`pkg2` は `pkg3` より約 2 倍のリソースを消費し、`pkg3` は `pkg1` の約 1.5 倍のリソースを消費するものとします。このような場合、パッケージ構成ファイルで次のように指定できます。

- `pkg1` について

```
weight_name package_limit
weight_value 2
```
- `pkg2` について

```
weight_name package_limit
weight_value 6
```
- `pkg3` について

```
weight_name package_limit
weight_value 3
```

これで、予約済みの `CAPACITY_VALUE package_limit` の `CAPACITY_VALUE` が 10 の `node1` は、一度にどの 2 つのパッケージでも実行できますが、3 つすべては実行できません。さらに、大きな方の 2 つのパッケージ `pkg2` と `pkg3` が `node1` で同時に実行されないようにする場合は、どちらか、または両方の `weight_value` を増やし、合計が 10 を超えるようにするか、`node1` のキャパシティを 8 に減らすこともできます。

留意すべき事項

以下の留意事項は、特に[単純な方法\(108 ページ\)](#)に適用されます。すべての重みとキャパシティに適用される[規則とガイドライン \(113 ページ\)](#)と合わせてお読みください。

- 予約済みの `CAPACITY_NAME package_limit` を使用する場合は、これがこのクラスターで定義できる唯一のキャパシティと重みになります。

- 予約済みの `CAPACITY_NAME package_limit` を使用する場合は、すべてのパッケージの重みは 1 です。このデフォルトは、上記の例に示すように、`weight_name` および `weight_value` パラメーターを使用して **パッケージ** 構成ファイルに定義することで変更することができます。
(パッケージ構成ファイルで別の重みを明示的に割り当てなかったパッケージについては、重みは、デフォルトの 1 のままです。)
- 予約済みの `CAPACITY_NAME package_limit` を使用する場合、`weight_name` を使用するのであれば、それも `package_limit` にする必要があります。
- すべてのノードについてキャパシティを定義する必要はありません。定義しなかった場合、ノードキャパシティは無制限であるとみなされ、使用可能なパッケージを同時にいくつでも実行することができます。
- 単一キャパシティのみを定義するが、デフォルトの重みは 1 ではなく 0 にしたい場合は、予約名の `package_limit` を使用しないでください。別の名前 (たとえば `resource_quantity`) を使用し、**包括的な方法**に従ってください。将来複数のキャパシティを使用する予定がある場合にも、その方法が適しています。

重みとキャパシティの構成の詳細については、[詳細情報\(114 ページ\)](#) にリストされているドキュメントを参照してください。

包括的な方法

単純な方法(108 ページ) がニーズに合わない場合は、こちらの方法を使用してください (先に進む前に、ここまでの項を読んでいることを確認してください)。包括的な方法は、パッケージが消費するコンピューティングリソースの量がさまざまに異なり、パッケージ間の単純な 1 対 1 の比較が実用的でない場合に有効です。

- ❶ **重要:** 2 つの方法を組み合わせることはできません。ノードで予約済みのキャパシティ `package_limit` を使用すると、クラスター内で他のタイプのキャパシティおよび重みを定義することはできなくなるため、その場合には **単純な方法** に限定されてしまいます。

キャパシティの定義

最初に、定義するキャパシティを決定します。クラスターに対し、最高 4 つの異なるキャパシティを定義できます。

キャパシティを定義するのに、「プロセッサ」、「メモリ」、「IO」などの日常的な名前を使用したいかもしれませんが、そうすべきではありません。実際、もしパッケージが、単純な名前前で表現するのは難しいさまざまな要素が競合する場合に、「プロセッサ」のような単純なリソース名で表現すると誤解を生みかねません。いずれにせよ、ノードにキャパシティとパッケージ重みを指定するために使用する、現実世界の名前の意味は、Serviceguard の関知するところではありません。Serviceguard は単に、ノードに対して構成されたキャパシティが、そのノード上で動作するパッケージの重みの合計を超えないようにするだけです。

たとえば、`CAPACITY_NAME` と `weight_name` で `processor` を定義し、さらに `CAPACITY_NAME` と `weight_name` で `memory` を定義し、ノードの `processor` キャパシティが 10 で `memory` キャパシティが 1000 の場合、Serviceguard は常に、ノードで実行されるパッケージの `processor` 重みの合計が 10 を超えず、`memory` 重みの合計も 1000 を超えないようにします。しかし、Serviceguard は現実世界の `processor` と `memory` の意味は感知しないため、実際のプロセッサおよびメモリ使用率とは何の対応もありません。apples や oranges という名前を使用しても結果は同じです。

たとえば、以下の構成を行うとします。

- 2 ノードクラスターで 4 つのパッケージを実行します。これらのパッケージはリソースを求めて競合しますが、それらのリソースを単に A と B と呼ぶことにします。
- node1 には、A については 80、B については 50 のキャパシティがあります。
- node2 には、A については 60、B については 70 のキャパシティがあります。

- pkg1 は、A のキャパシティを 60、B のキャパシティを 15 使用します。
- pkg2 は、A のキャパシティを 40、B のキャパシティを 15 使用します。
- pkg3 は、A のキャパシティをほんのわずか (0)、B のキャパシティを 35 使用します。
- pkg4 は、A のキャパシティを 20、B のキャパシティを 40 使用します。

pkg1 と pkg2 の 2 つで、A のキャパシティが 100、B のキャパシティが 30 が必要です。したがって、pkg1 と pkg2 は、どちらのノードでも一緒に実行することはできません。どちらのノードも、B についてはこれら 2 つのパッケージを同時に実行するのに十分なキャパシティがあるのですが、A のキャパシティは十分ではありません。

pkg3 と pkg4 の 2 つで、A のキャパシティが 20、B のキャパシティが 75 が必要です。したがって、pkg3 と pkg4 は、どちらのノードでも一緒に実行することはできません。どちらのノードも、A についてはこれら 2 つのパッケージを同時に実行するのに十分なキャパシティがあるのですが、B のキャパシティは十分ではありません。

例 2

これらのキャパシティを定義し、個々のノードの制限を設定するには、クラスター構成ファイルで次のようなエントリーを作成します。

```
CLUSTER_NAME cluster_23
...
NODE_NAME node1
...
CAPACITY_NAME A
CAPACITY_VALUE 80
CAPACITY_NAME B
CAPACITY_VALUE 50
NODE_NAME node2
CAPACITY_NAME A
CAPACITY_VALUE 60
CAPACITY_NAME B
CAPACITY_VALUE 70
...
```

注記: クラスター内のすべてのノードのキャパシティを定義する必要はありません。どのノードについてもキャパシティを定義しなかった場合、ノードには該当のキャパシティが無限大あるものとみなされます。この例では、特定のノードのキャパシティAを定義しなかった場合、自動的に、pkg1 と pkg2 のパッケージにどれだけのA 重みを割り当てたかに関係なく、この 2 つのパッケージをこのノードで同時に実行できることになります。キャパシティB を定義しなかった場合は、pkg3 と pkg4 を同時に実行できることになり、キャパシティ A、キャパシティ B のどちらも定義しなかった場合、4 つのパッケージをすべて同時にこのノードで実行できることになります。

ノードのキャパシティを定義したら、次の手順ではパッケージの重みを構成します。「[重みの定義](#)」を参照してください。

重みの定義

パッケージの重みはノードのキャパシティに対応しており、どのキャパシティ/重みのペアについても、CAPACITY_NAME と weight_name が一致している必要があります。

パッケージ構成ファイルで個々のパッケージの重みを定義しますが、特定の重みについてクラスター全体のデフォルト値を定義することもできます。また、クラスター全体のデフォルト値

を定義した場合、このデフォルト値は、パッケージ構成ファイルで明示的に変更されない限り、すべてのパッケージの重みを指定します。

注記: 唯一の例外として、システムマルチノードパッケージには重みを定義できないため、クラスター全体のデフォルト重みはシステムマルチノードパッケージには適用されません。

デフォルトの重みの定義

「[キャパシティの定義](#)」(110 ページ)以降のサンプルを進めるにあたり、pkg1 と pkg2 以外のすべてのパッケージではほぼ同量のキャパシティA を使用し、pkg3 と pkg4 以外のすべてのパッケージではほぼ同量のキャパシティB を使用するものとしましょう。クラスター構成ファイルで `WEIGHT_DEFAULT` パラメーターを使い、以下のように両方の重みのデフォルト値を設定することができます。

例 3

```
WEIGHT_NAME A
WEIGHT_DEFAULT 20
WEIGHT_NAME B
WEIGHT_DEFAULT 15
```

この場合、パッケージ構成ファイルで重みA が定義されていないパッケージはすべて、重みA が 20 になり、パッケージ構成ファイルで重みB が定義されていないパッケージはすべて、重みB が 15 になります。

クラスター構成ファイルで先に定義したキャパシティの場合(「[キャパシティの定義](#)」を参照)、node1 では、A と B の両方についてデフォルト値を使用する 3 つのパッケージをどれでも実行することができます。この結果、このノード上で 20 単位のA キャパシティと 5 単位のB キャパシティの予備が生まれます。

個々のパッケージの重みの定義

クラスター構成ファイルで定義するキャパシティごとに(「[キャパシティの定義](#)」を参照)、特定のパッケージに対応する重みを割り当てるにあたり、次のような選択肢があります。

1. クラスター全体のデフォルトの重みを構成し、パッケージにこのデフォルト値を使用させる。
2. クラスター全体のデフォルトの重みを構成するが、パッケージ構成ファイルでこのパッケージの重みを上書きする。
3. クラスター全体のデフォルトの重みは構成しないが、パッケージ構成ファイルでこのパッケージに重みを割り当てる。
4. クラスター全体のデフォルトの重みは構成せず、パッケージ構成ファイルでこのパッケージに重みを割り当てることもしない。

注記: オプション 4 の場合、この特定のキャパシティに関する限りパッケージに「重みがない」ことになり、他のパッケージがこのキャパシティを完全に消費しているノード上でも実行できます。

(クラスター全体のデフォルトの重みを定義した場合でも、特定のキャパシティについて 1 つのパッケージを「重みなし」にすることができます。そのパッケージのクラスター構成ファイルで対応する重みを単に 0 に設定するだけです。)

「[キャパシティの定義](#)」(110 ページ)以降の例を進める場合、オプション 1 と 2 を使用して pkg1 から pkg4 までの重みを設定することができます。

例 4

```
pkg1 のパッケージ構成ファイルでは
weight_name A
weight_value 60
```

```
pkg2 のパッケージ構成ファイルでは
weight_name A
weight_value 40
pkg3 のパッケージ構成ファイルでは
weight_name B
weight_value 35
weight_name A
weight_value 0
pkg4 のパッケージ構成ファイルでは
weight_name B
weight_value 40
```

- ❗ **重要:** パッケージ構成ファイルの `weight_name` は、クラスター構成ファイルの対応する `CAPACITY_NAME` と正確に一致していなければなりません。これはスベルだけでなく大文字小文字の区別にも適用されます。`weight_name a` は、`CAPACITY_NAME A` とは一致しません。対応するキャパシティが定義されていない限り、重みを定義することはできません。パッケージ構成ファイル内の重みを定義したのに、パッケージの `node_name` リスト(164 ページ)のどのノードでも対応するキャパシティがクラスター構成ファイルに指定されていなかった場合、またはクラスター構成ファイルでデフォルトの重みを定義したのに、クラスター内のどのノードにも同じ名前のキャパシティが指定されていない場合、`cmapplyconf` は失敗します。

この例で留意すべき事項

- pkg1 または pkg2 についてはB 重みを構成しなかったため、これらのパッケージには例 3 (112 ページ) でクラスター構成ファイルに設定したデフォルトのB の重み (15) が設定されます。同様に、pkg4 にはデフォルトのA の重み (20) が設定されます。
- pkg3 については、B の重みの 35 を構成しましたが、A の重みは構成していません。
- pkg1 は、node2 のA キャパシティのすべてを消費します。pkg1 がこのノードで実行されている間、A の重みを持つ他のパッケージはこのノード上では実行できません。
しかし、node2 では pkg1 の実行中も pkg3 を実行できます。pkg3 にはA の重みがなく、pkg1 は node2 のB キャパシティの 15 単位 (デフォルト値) しか消費していないからです。この結果、pkg3 は 35 単位を消費できることになります (B の重みを持つ他のパッケージがすでにこのノードで実行中でないと仮定して)。
- 同様に、A の重みを持つパッケージが node2 ですでに実行されていれば、pkg1 はそのノードで起動することはできません (他のパッケージを強制的に移動させるだけの優先順位が pkg1 がない限り。「[パッケージ重みとパッケージの優先順位および依存関係との相互作用](#)」(114 ページ)を参照してください)。このことは、パッケージがノード上の対応するキャパシティの使用可能な量を超える重みを持っている場合、常にあてはまります。

規則とガイドライン

以下の規則とガイドラインは、キャパシティと重みを構成するための[単純な方法](#) (108 ページ) と[包括的な方法](#) (110 ページ) の両方に適用されます。

- クラスター全体で最大 4 つのキャパシティと、それに対応する重みを定義できます。

注記: ただし、予約済みの `CAPACITY_NAME package_limit` を使用する場合、その 1 つのキャパシティと対応する重みしか定義できません。「[単純な方法](#)」(108 ページ)を参照してください。

- ノードのキャパシティは、**クラスター**構成ファイルで `CAPACITY_NAME` および `CAPACITY_VALUE` パラメーターを使って定義します。

- キャパシティは、クラスターの稼働中に追加、変更、または削除することができます。そのため、一部のパッケージが移動されたり、停止されて再起動されないこともあります。
- パッケージの重みは、**クラスター**構成ファイルで `WEIGHT_NAME` および `WEIGHT_DEFAULT` パラメーターを使って定義するか、**パッケージ**構成ファイルで `weight_name` および `weight_value` パラメーターを使って定義できます。両方で定義することもできます。
- 重みは、[failover_policy\(166 ページ\)](#) が `configured_node` で [failback_policy\(167 ページ\)](#) が `manual` のマルチノードパッケージとフェイルオーバーパッケージにのみ割り当て (および `WEIGHT_DEFAULT` は適用する) ことができます。
- パッケージに重み (`weight_name` と `weight_value`) を定義する場合、パッケージの `node_name` リスト ([164 ページ](#)) の少なくとも1つのノードに対して、必ず対応するキャパシティもクラスター構成ファイルで定義してください (`CAPACITY_NAME` および `CAPACITY_VALUE`)。そうしないと、パッケージを適用しようとする `cmapplyconf` は失敗します。
- 重み (クラスター全体の `WEIGHT_DEFAULT` とパッケージ構成ファイルで定義された重みの両方) は、クラスターが稼働中で、パッケージが実行中に変更することができます。そのため、一部のパッケージが移動されたり、停止されて再起動されないこともあります。

詳細情報

キャパシティについての詳細は、以下の `CAPACITY_NAME` および `CAPACITY_VALUE` のコメントを参照してください。

- クラスター構成ファイル
- `cmquerycl (1m)` のマンページ
- 本書の「[クラスター構成のパラメーター](#)」 ([82 ページ](#)) の項

重みについての詳細は、以下の `weight_name` および `weight_value` のコメントを参照してください。

- パッケージ構成ファイル
- `cmmakepkg (1m)` のマンページ
- 本書の「[パッケージのパラメーターの説明](#)」 ([162 ページ](#)) の項

詳細な説明や使用例については、ホワイトペーパー『Using Serviceguard's Node Capacity and Package Weight』を参照してください。このホワイトペーパーは、<http://www.hp.com/go/linux-serviceguard-docs> -> [White papers] にあります。

パッケージ重みとパッケージの優先順位および依存関係との相互作用

必要に応じて、Serviceguard は、重みを持つ優先順位の高いパッケージに譲るために、重みを持つ優先順位の低い動作中のパッケージを停止します。ただし、優先順位のない動作中のパッケージ (つまり、`priority` がデフォルトの `no_priority` に設定されている) が、優先順位のない停止中のパッケージに譲るために停止されることはありません。優先順位のない停止中の2つのパッケージを、それらのパッケージの重みを支えるだけのノードキャパシティがないため両方を起動できない場合、Serviceguard はどちらのパッケージを起動するかを判定します。

例 1

- `pkg1` はノード `turkey` と `griffon` で実行されるように構成されています。重みは 1 で優先順位は 10 です。停止中で、切り替えは無効になっています。
- `pkg2` はノード `turkey` と `griffon` で実行されるように構成されています。重みは 1 で優先順位は 20 です。ノード `turkey` で実行中であり、切り替えは有効になっています。
- `turkey` と `griffon` では、それぞれ 1 つのパッケージを実行できます (`package_limit` は 1 に設定されています)。

pkg1 の切り替えを有効にした場合、Serviceguard は turkey 上で優先順位の低い pkg2 を停止します。次に turkey で pkg1 を起動し、griffon で pkg2 を再起動します。

pkg1 にも pkg2 にも優先順位が設定されていない場合、pkg2 は turkey で実行を続け、pkg1 は griffon で実行されます。

例 2

- pkg1 はノード turkey と griffon で実行されるように構成されています。重みは 1 で優先順位は 10 です。ノード turkey で実行中であり、切り替えは有効になっています。
- pkg2 はノード turkey と griffon で実行されるように構成されています。重みは 1 で優先順位は 20 です。ノード turkey で実行中であり、切り替えは有効になっています。
- pkg3 はノード turkey と griffon で実行されるように構成されています。重みは 1 で優先順位は 30 です。停止中で、切り替えは無効になっています。
- pkg3 には、pkg2 に対し same_node 依存関係があります。
- turkey と griffon では、それぞれ 2 つのパッケージを実行できます (package_limit は 2 に設定されています)。

pkg3 の切り替えを有効にしても、このパッケージが依存するパッケージ pkg2 が、すでに 2 つのパッケージ(キャパシティの限界)を実行しているノード turkey 上で実行中であるため、pkg3 は停止したままです。pkg3 は pkg2 よりも優先順位が低いので、この 2 つのパッケージを実行できる griffon に pkg2 をドラッグすることはできません。

外部スクリプトについて

Serviceguard A.11.18 では、モジュラーパッケージのパッケージ構成テンプレートにより、外部スクリプトを明示的に指定できます。このスクリプトは従来のスクリプトの CUSTOMER_DEFINED_FUNCTION に代わり、以下のいずれかの時点で実行できます。

- パッケージの起動時とシャットダウン時。つまり基本的には、パッケージが実行する最初と最後の関数です。このスクリプトは、パラメーター `external_pre_script` (178 ページ) により起動されます。**または**
- パッケージの実行中、ボリュームグループとファイルシステムの起動後、IP アドレスの割り当て後、かつサービスおよびリソースの関数の実行前。そして再度、逆順でパッケージのシャットダウン時。このスクリプトは、パラメーター `external_script` (178 ページ) により起動されます。

このスクリプトは、cmcheckconf と cmapapplyconf でパッケージを検証する際にも実行されます。

1 つのパッケージは両方の種類のスクリプトを使うことができ、それぞれの種類のスクリプトを複数起動することができます。この場合、スクリプトは、パッケージ構成ファイルにリストされている順序 (パッケージのシャットダウン時には逆順) で実行されます。

場合によっては、外部スクリプトを使用しているパッケージの実行中に、その外部スクリプトを名前変更したり、入れ替えたりすることができます。「稼働中のパッケージで使用される外部スクリプトの名前変更または交換」(226 ページ) を参照してください。

外部スクリプトには、3 つのエントリーポイント start、stop、および validate が必要で、以下の値のいずれかで終了しなければなりません。

- 0 - 成功を示します。
- 1 - スクリプトが失敗したため、パッケージが停止されるが、再起動してはならないことを示します。
- 2 - パッケージが別のノードで再起動されるか、使用可能な他のノードがない場合は停止されることを示します。

注記: validate エントリーポイントの場合、終了値 1 と 2 は同様に扱われます。どちらの値でも検証の失敗を示すことができます。

スクリプトは、パッケージを実行するパッケージマネージャーまたはマスター制御スクリプトがエクスポートした環境変数の標準セット (パッケージ名 `SG_PACKAGE` と、ローカルノード名 `SG_NODE` を含む) を使うことができます。また、ログ関数や他のユーティリティ関数を取り込むための関数を呼び出すこともできます。このような関数の 1 つ、`sg_source_pkg_env()` を呼び出すと、`pev_` パラメーター(178 ページ)により構成された、パッケージ固有の環境変数を含め、このパッケージに構成されているすべてのパラメーターにアクセスできます。

注記: `SG_PACKAGE` や `SG_NODE` など一部のパラメーターは、パッケージの実行時および停止時にのみ使用できます。パッケージの検証時には使用できません。検証時には `SG_PACKAGE` の代わりに `SG_PACKAGE_NAME` を使用できます。

詳細は、`$SGCONF/examples/external_script.template` のテンプレートを参照してください。

以下に、サンプルのスクリプトを示します。このスクリプトでは、一部のアプリケーションを監視する Serviceguard サービスとして構成される、`monitor.sh` という別のスクリプトがあることを前提としています。`monitor.sh` スクリプト (ここには記載されていません) は、パッケージ構成ファイルで定義されている `PEV_MONITORING_INTERVAL` パラメーターを使って、監視対象のアプリケーションを定期的にポーリングします。例を次に示します。

`PEV_MONITORING_INTERVAL 60`

サンプルスクリプトは検証時に、`PEV_MONITORING_INTERVAL` パラメーターと監視サービスが適切に構成されていることを確認します。また、このスクリプトは起動時と停止時に、この時間間隔をログファイルに出力します。

```
#!/bin/sh
# Source utility functions.
if [[ -z $SG_UTILS ]]
then
    . $SGCONF.conf
    SG_UTILS=$SGCONF/scripts/mscripts/utils.sh
fi

if [[ -f ${SG_UTILS} ]]; then
    . ${SG_UTILS}
    if (( $? != 0 ))
    then
        echo "ERROR: Unable to source package utility functions file: ${SG_UTILS}"
        exit 1
    fi
else
    echo "ERROR: Unable to find package utility functions file: ${SG_UTILS}"
    exit 1
fi

# Get the environment for this package through utility function
# sg_source_pkg_env().
sg_source_pkg_env $*

function validate_command
{
    typeset -i ret=0
    typeset -i i=0
    typeset -i found=0
    # check PEV_ attribute is configured and within limits
    if [[ -z PEV_MONITORING_INTERVAL ]]
    then
        sg_log 0 "ERROR: PEV_MONITORING_INTERVAL attribute not configured!"
        ret=1
    elif (( PEV_MONITORING_INTERVAL < 1 ))
    then
        sg_log 0 "ERROR: PEV_MONITORING_INTERVAL value ($PEV_MONITORING_INTERVAL) not within legal limits!"
        ret=1
    fi
    # check monitoring service we are expecting for this package is configured
    while (( i < ${#SG_SERVICE_NAME[*]} ))
    do
        case ${SG_SERVICE_CMD[i]} in
            *monitor.sh*) # found our script
                found=1
                break
                ;;
            *)
                ;;
        esac
        (( i = i + 1 ))
    done
    if (( found == 0 ))
```

```

    then
        sg_log 0 "ERROR: monitoring service not configured!"
        ret=1
    fi
    if (( ret == 1 ))
    then
        sg_log 0 "Script validation for $SG_PACKAGE_NAME failed!"
    fi
    return $ret
}

function start_command
{
    sg_log 5 "start_command"

    # log current PEV_MONITORING_INTERVAL value, PEV_ attribute can be changed
    # while the package is running
    sg_log 0 "PEV_MONITORING_INTERVAL for $SG_PACKAGE_NAME is $PEV_MONITORING_INTERVAL"
    return 0
}

function stop_command
{
    sg_log 5 "stop_command"
    # log current PEV_MONITORING_INTERVAL value, PEV_ attribute can be changed
    # while the package is running
    sg_log 0 "PEV_MONITORING_INTERVAL for $SG_PACKAGE_NAME is $PEV_MONITORING_INTERVAL"
    return 0
}

typeset -i exit_val=0
case ${1} in
    start)
        start_command $*
        exit_val=$?
        ;;
    stop)
        stop_command $*
        exit_val=$?
        ;;
    validate)
        validate_command $*
        exit_val=$?
        ;;
    *)
        sg_log 0 "Unknown entry point $1"
        ;;
esac
exit $exit_val

```

外部スクリプトでの Serviceguard コマンドの使用

外部スクリプトで、Serviceguard コマンド (cmmodpkg など) を使うことができます。ただし、これらのコマンドで、パッケージそのもの (すなわちその外部スクリプトを実行するパッケージ) とデータをやり取りしてはなりませんが、他のパッケージとはやり取りすることができます。ただし、このやり取りは慎重にコーディングしてください。

Serviceguard コマンドが他のパッケージとデータをやり取りする場合は、コマンドループが発生しないように注意してください。コマンドループが発生するのは次のような場合です。pkg1 スクリプトが pkg2 の cmmodpkg -d を実行し、pkg2 スクリプトが pkg1 の cmmodpkg -d を実行するとします。pkg1 と pkg2 が同時に起動されると、pkg1 スクリプトは pkg2 に対して cmmodpkg を実行しようとし、ところがこの cmmodpkg は、pkg2 の起動が完了しなければ実行できません。一方 pkg2 スクリプトも pkg1 に対して cmmodpkg を実行しようとし、同様に、pkg2 は pkg1 の起動が完了しなければ実行できないので、コマンドループが発生します。

このようなコマンドループが発生しないように、すべてのパッケージ (特に、外部スクリプトで Serviceguard コマンドを使うパッケージ) に run_script_timeout と halt_script_timeout を指定しておくといでしょう。タイムアウトの指定をしていないときにパッケージで上記のようなコマンドループが発生すると、クラスターがハングするなど、望ましくない状況が発生する可能性があります。

パッケージがシャットダウンした原因の調査

外部スクリプト (従来のパッケージ制御スクリプトの CUSTOMER_DEFINED_FUNCTIONS 領域) を使うと、パッケージがシャットダウンした原因を調べることができます。

パッケージが停止すると、Serviceguard はパッケージ制御スクリプト内の環境変数 `SG_HALT_REASON` に、以下の値のいずれかを設定します。

- `failure` - パッケージが依存しているサブネット、リソース、またはサービスの障害によりパッケージが停止した場合に設定されます。
- `user_halt` - `cmhaltpkg` コマンドまたは `cmhaltnode` コマンド、あるいはそれに相当する Serviceguard Manager の動作によってパッケージが停止した場合に設定されます。
- `automatic_halt` - 依存対象パッケージの障害によりパッケージが自動的にフェイルオーバーしたか、パッケージがプライマリノードに自動的にフェイルバックした (`failback_policy = automatic`) 場合に設定されます。

この変数を問い合わせるカスタムコードをパッケージに追加して、パッケージが停止した原因を調べ、適切に対処することができます。従来のパッケージの場合は、パッケージ制御スクリプトの `CUSTOMER_DEFINED_FUNCTIONS` 領域にある `customer_defined_halt_cmds()` 関数にコードを追加します (「[パッケージ制御スクリプトへのユーザー定義関数の追加](#)」(222 ページ) を参照)。モジュラーパッケージの場合は、パッケージの外部スクリプトにコードを追加します (「[外部スクリプトについて](#)」(115 ページ) を参照)。

たとえば、データベースパッケージが管理者によって停止されている (`SG_HALT_REASON` に `user_halt` が設定されている) 場合は、カスタムコードでデータベースを正常にシャットダウンすることが考えられます。一方、`SG_HALT_REASON` に、パッケージが異常停止した (たとえば、パッケージの依存対象サービスの障害などによる) ことを示す `failure` が設定されている場合は、強制シャットダウンが必要となることがあります。

last_halt_failed フラグ

`cmviewcl -v -f line` では、`last_halt_failed` フラグが表示されます。

注記: `last_halt_failed` は、`cmviewcl` の行形式のみで表示され、デフォルトの表形式では表示されません。これを表示するには、`-f line` オプションを使用します。

`last_halt_failed` の値は、停止スクリプトを正常に終了した場合、ノードをクラスターに追加して以来実行していない場合、またはパッケージをノードで実行するよう構成して以来実行していない場合、`no` が設定されます。それ以外の場合は、`yes` です。

クロスサブネットフェイルオーバーについて

ルーターで結合された複数のサブネットにまたがったクラスターを構成することができます。この構成では、一部のノードがあるサブネットを利用し、別のノードが別のサブネットを利用します。これを、クロスサブネット構成といいます (「[クロスサブネット構成](#)」(25 ページ) を参照)。このような環境では、あるサブネット上のノードから別のサブネット上のノードにフェイルオーバーするようにパッケージを構成できます。

クロスサブネットのフェイルオーバー用にパッケージを構成する場合は、以下の付帯事項があります。

- モジュラーパッケージでは、サブネットにまたがったパッケージのフェイルオーバーを可能にするために、パッケージ構成ファイルで以下の 2 つの新しいパラメーターを設定する必要があります。
 - `ip_subnet_node` (171 ページ) - サブネットが構成されているノードを示します。
 - `monitored_subnet_access` (169 ページ) - 監視対象サブネットがすべてのノード上で構成されているか (FULL)、一部のノードでのみ構成されているか (PARTIAL) を示します (監視対象サブネットの `monitored_subnet_access` が未構成のままの場合は、FULL と同等です)。

(従来のパッケージについては、「[クロスサブネットフェイルオーバーの構成](#)」(224 ページ) を参照してください。)

- パッケージ構成ファイルで `node_name` にワイルドカード (*) を使用することはできません。ワイルドカードを使用すると、同じサブネットのノードへのフェイルオーバーが可能であっても、パッケージがサブネットを越えてフェイルオーバーする可能性があります。サブネットを越えるフェイルオーバーは、同じサブネットでのフェイルオーバーよりも時間がかかる場合があります。ノードは、ワイルドカードを使用せずに、優先順位の高い順にリストしてください。
- この環境でアプリケーションを配備する際には、慎重に検討する必要があります。「[アプリケーションの配備への影響](#)」(119 ページ) を参照してください。
- `monitored_subnet` に対し、パッケージの構成ファイルで、`monitored_subnet_access` に `PARTIAL` が設定されている場合、そのパッケージの `node_name` リスト(164 ページ) 内の 1 台以上のノードでそのサブネットが構成されている必要があります。
逆に、このパッケージに対して監視されるすべてのサブネットで `PARTIAL` アクセスが構成されている場合、`node_name` リストの各ノードでは、これらのサブネットのうち 1 つ以上が構成されている必要があります。
 - 他のクラスター構成と同様に、ノード上に構成されて、パッケージ構成ファイル中で監視対象サブネットとして指定されているサブネットが動作状態でない場合、パッケージはノード上で開始されません。

アプリケーションの配備への影響

再配置可能 IP アドレスは、パッケージが別のサブネット上のノードにフェイルオーバーしたときに変わるため、以下の事項を確認しておく必要があります。

- パッケージが使っていたホスト名が、新しい再配置可能 IP アドレスに正しく再マッピングされる。
- パッケージが実行するアプリケーションは、クライアントがパッケージの新しい再配置可能 IP アドレスに再接続できるように構成されていなければならない。
最悪の場合 (アプリケーションが動作していたサーバーがダウンしているとき)、クライアントは TCP の `tcp_timeout` 時間 (一般的に約 10 分) が経過するまで古い IP アドレスでリトライし続けた後に障害を検出し、接続をリセットすることがあります。

詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] にあるホワイトペーパー『Technical Considerations for Creating a Serviceguard Cluster that Spans Multiple IP Subnets』を参照してください。

サブネットを越えてフェイルオーバーするパッケージの構成: 例

サブネットを越えてフェイルオーバーするようにパッケージを構成するためには、パッケージ構成ファイルに追加の編集を行う必要があります。

注記: ここでは、モジュラーパッケージの例を説明します。従来のパッケージについては、「[クロスサブネットフェイルオーバーの構成](#)」(224 ページ) を参照してください。

パッケージ `pkg1` を、NodeA、NodeB、NodeC、および NodeD からなるクラスター内のすべてのノード間でフェイルオーバーできるように構成するとします。

NodeA と NodeB は、NodeC と NodeD が使わないサブネット 15.244.65.0 を使います。
NodeC と NodeD は、NodeA と NodeB が使わないサブネット 15.244.56.0 を使います。
(`cmquerycl` の出力例は、「[クロスサブネット情報の取得](#)」(144 ページ) を参照してください。)

`node_name` の構成

まず、`pkg1` が他のサブネット上のノードにフェイルオーバーするのは、やむを得ない場合だけであることを確認する必要があります。たとえば、このパッケージが NodeA 上で実行されていて、フェイルオーバーする必要がある場合、クロスサブネットのオーバーヘッドがある

NodeC や NodeD へのフェイルオーバーを行うのは、同じサブネット上の NodeB へのフェイルオーバーを試みた後です。

nodeA が pkg1 のプライマリノード (そのパッケージが通常起動されるノード) だとすると、パッケージ構成ファイルに次のような `node_name` エントリーを作成します。

```
node_name nodeA
node_name nodeB
node_name nodeC
node_name nodeD
```

monitored_subnet_access の構成

pkg1 が実行されている場所に応じてサブネット 15.244.65.0 または 15.244.56.0 を監視するには、pkg1 のパッケージ構成ファイルの `monitored_subnet` と `monitored_subnet_access` を、次のように構成します。

```
monitored_subnet 15.244.65.0
monitored_subnet_access PARTIAL
monitored_subnet 15.244.56.0
monitored_subnet_access PARTIAL
```

注記: いずれのサブネットもすべてのノード上で利用可能というわけではないため、これらのサブネットのいずれかに対して `monitored_subnet_access` を FULL として構成する (または、`monitored_subnet_access` を設定しない) と、パッケージの構成が失敗します。

ip_subnet_node の構成

次に、どのノード上にどのサブネットを構成するかを指定する必要があります。この例では、パッケージ構成ファイル内の次のようなエントリーを使って指定します。

```
ip_subnet 15.244.65.0
ip_subnet_node nodeA
ip_subnet_node nodeB
ip_address 15.244.65.82
ip_address 15.244.65.83
ip_subnet 15.244.56.0
ip_subnet_node nodeC
ip_subnet_node nodeD
ip_address 15.244.56.100
ip_address 15.244.56.101
```

パッケージの構成: 次の手順

パッケージの構成を開始できるようになったら、[第6章「パッケージとサービスの構成」](#) (157 ページ) に進みます。「[パッケージモジュールの選択](#)」 (157 ページ) から開始します。(必要に応じて、パッケージごとに別のワークシートを使って、あらかじめパッケージ構成データを収集しておくことができます。[付録 C](#) には未記入のワークシートがあります。)

クラスター規模の変更の計画

オンラインで (クラスター稼働中に) ノードを追加する予定がある場合は、追加するノードが他のクラスターノードと同じハートビートサブネット、同じロックディスクに接続されるようにしなければなりません。

クラスターロックの構成を選択する際には、クラスターノードの追加が必要になる可能性についても考慮してください。2 ノードのクラスターにはクラスターロックを使用する**必要**があり、4 ノードを超えるクラスターにはロック LUN を**使用せず**、クォーラムサーバーを使用します。したがって、最終的に 5 ノードが必要になる場合は、クォーラムサーバーを使用して初期構成を構築してください。

クラスターの稼働時にクラスター構成からノードを削除する場合は、削除後のクラスター構成でも前述のクラスターロックの規則に準拠する必要があります。詳細は、「[クラスターロックのプランニング](#)」(77 ページ) を参照してください。

ノードをオンラインで追加する予定があり、新しいノードでパッケージを実行する場合は、クラスター内にあるパッケージの既存のボリュームグループが新しいノードにインポートされていることを確認してください。また、`MAX_CONFIGURED_PACKAGES` パラメーターは、使用するパッケージの総数に十分対応できるように設定してください。「[クラスター構成のパラメーター](#)」(82 ページ) を参照してください。

5 HA クラスターの構成

本章と次章では、Serviceguard クラスターのセットアップに必要な構成作業について説明します。これらの作業は構成ノードという 1 つのノードで実行し、その結果生成されるバイナリファイルが、Serviceguard によりクラスター内のすべてのノードに配布されます。本章の例では、構成ノードを `ftsys9`、ターゲットノードを `ftsys10` としています。

本章では、以下の主要項目について説明します。

- システムの準備
- クラスターの構成 (141 ページ)
- 稼働中のクラスターの管理 (152 ページ)

パッケージの構成については、次章で説明します。

各コマンドの構文と使用方法についての詳細は、Serviceguard の各コマンドのマニュアルを参照してください。

システムの準備

クラスターを構成する前に、すべてのクラスターノードに Serviceguard がインストールされており、そのすべてのノードに、適切なセキュリティファイル、カーネル構成、NTP (network time protocol) 構成があることを確認してください。

Serviceguard のインストールとアップデート

Serviceguard のインストールおよびアップデートの詳細は、<http://www.hp.com/go/linux-serviceguard-docs> -> [Release Notes] で、使用しているバージョンのリリースノートを参照してください。

Serviceguard のファイルの位置

Serviceguard では、`/etc/cmcluster.conf` という特別なファイルを使って、Linux ファイルシステム内の構成ファイルおよびログファイルの位置を定義します。ディストリビューションごとにファイルの位置は異なります。以下に、Red Hat ディストリビューションの例を示します。

```
##### cmcluster.conf #####
#
# Highly Available Cluster file locations
#
# This file must not be edited
#####
SGROOT=/usr/local/cmcluster      # SG root directory
SGCONF=/usr/local/cmcluster/conf  # configuration files
SGSBIN=/usr/local/cmcluster/bin   # binaries
SGLBIN=/usr/local/cmcluster/bin   # binaries
SGLIB=/usr/local/cmcluster/lib    # libraries
SGRUN=/usr/local/cmcluster/run    # location of core dumps from daemons
SGAUTOSTART=/usr/local/cmcluster/conf/cmcluster.rc # SG Autostart file
```

以下に、SUSE ディストリビューションの例を示します。

```
##### cmcluster.conf #####
#
# Highly Available Cluster file locations
#
# This file must not be edited
#####
SGROOT=/opt/cmcluster            # SG root directory
SGCONF=/opt/cmcluster/conf       # configuration files
SGSBIN=/opt/cmcluster/bin        # binaries
SGLBIN=/opt/cmcluster/bin        # binaries
```

```

SGLIB=/opt/cmcluster/lib          # libraries
SGRUN=/opt/cmcluster/run          # location of core dumps from daemons
SGAUTOSTART=/opt/cmcluster/conf/cmcluster.rc # SG Autostart file

```

本書では、システムファイル名にはこれらのパスを示す接頭辞が付いています。したがって、`$SGCONF/<FileName>` への参照は、このファイル内の定義を使って解決されます。たとえば、SGCONF が `/usr/local/cmcluster/conf` と定義されている場合、`$SGCONF/cmclconfig` ファイルの完全なパス名は、`/usr/local/cmcluster/conf/cmclconfig` です。

Serviceguard のコマンドアクセスの有効化

Serviceguard 構成を作成できるようにするには、Serviceguard コマンドを実行する前に**すべてのクラスターノード上で**以下の手順を実行する必要があります。

1. 必ず、ルートユーザーのパスに Serviceguard の実行可能ファイルを含めます。Red Hat の場合には、次の環境変数定義をルートユーザーのプロファイルに追加します。

```
PATH=$PATH:/usr/local/cmcluster/bin
```

SUSE の場合:

```
PATH=$PATH:/opt/cmcluster/bin
```

2. `/etc/man.config` ファイルを編集し、Red Hat の場合には次の行を含めます。

```
MANPATH /usr/local/cmcluster/doc/man
```

SUSE の場合:

```
MANPATH /opt/cmcluster/doc/man
```

これで、Serviceguard のマンページを使用できるようになります。

3. Serviceguard の変数を使用できるようにします。
システムで Serviceguard の変数が定義されていない場合、ユーザー root のログインプロファイルに `/etc/cmcluster.conf` ファイルを取り込んでください。

```
. /etc/cmcluster.conf
```

次に示すように、変数へのアクセスが可能になったことを確認してください。

```
cd $SGCONF
```

ルートレベルアクセスの構成

以降では、今後利用するクラスター内にノード間のルートアクセスを設定する方法について説明します。(クラスターの構成に進むと、さまざまなレベルの非ルートアクセスも定義します。「[クラスターへのアクセスの制御](#)」(146 ページ)を参照してください。)

注記: 詳細情報とアドバイスについては、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] -> [White Papers] にあるホワイトペーパー『Securing Serviceguard』を参照してください。

未構成ノードへのルートアクセスの許可

システムをクラスターに組み込み可能にするためには、クラスターに組み込む他のすべてのノードのルートユーザーによる、そのシステムに対する Linux ルートアクセスを有効にしなければなりません。Serviceguard では、`$SGCONF/cmclnodelist` ファイルというメカニズムでこの処理を行います。このファイルは、Serviceguard が最初にノードをクラスターに構成するときだけ参照されるため、「ブートストラップ」ファイルとも呼ばれます。このファイルは、その後は無視されます。デフォルトでは存在しないので、作成する必要があります。

このファイルの先頭には、次のようなコメントを追加することをお勧めします。

```
#####
```

```
# Do not edit this file!
```

```
# Serviceguard uses this file only to authorize access to an
# unconfigured node. Once the node is configured,
# Serviceguard will not consult this file.
```

#####

cmclnodelist 内のエントリーの形式は、次のとおりです。

```
[hostname] [user] [#Comment]
```

例:

```
gryf root      #cluster1, node1
sly  root      #cluster1, node2
bit  root      #cluster1, node3
```

この例では、この cmclnodelist ファイルが存在するノードへのルートアクセスが、ノード gryf、sly、および bit 上のルートユーザーに許可されます。

Serviceguard は、cmclnodelist ファイルでの「+」の使用も認めています。この指定は、任意の Serviceguard ノード上のルートユーザーがこのノード上の Serviceguard を構成できることを示します。

-
- ❶ **重要:** \$SGCONF/cmclnodelist が存在しないと、Serviceguard は ~/.rhosts を参照します。cmclnodelist を使うことを強くお勧めします。
-

注記: A.11.15 またはそれ以前のバージョンからクラスターをアップグレードするときには、\$SGCONF/cmclnodelist のエントリーは、クラスター構成ファイル内のアクセス制御ポリシーに自動的に変換されます。非ルートのユーザー名とホスト名の組には、すべてモニターロールが割り当てられます。

他のノードのルートユーザーが認識されるようにする

クラスターノード上の Linux ルートユーザーであれば、クラスターを構成できます。これには、あるノードの Serviceguard が別ノードのルートユーザーを認識できることが必要です。

Serviceguard は、identd デーモンを使ってユーザー名を検証し、ルートユーザーの場合は、identd がユーザー名 root を返した場合にのみ検証が成功します。identd は、最初に一致した UID 0 のユーザー名を返すため、クラスターに組み込む各ノードの /etc/passwd をチェックして、root ユーザーのエントリーが、UID 0 の他のエントリーより前に記述されていることを確認する必要があります。

identd について

ユーザー確認には identd の使用を強くお勧めします。したがって、今後利用する各クラスターノードは、このデーモンを実行するように構成してください。通常 identd は、/etc/init.d/xinetd から起動されます。

(お勧めできませんが、identd を無効にすることもできます。何らかの理由で identd を無効にしなければならない場合は、「[identd の無効化](#)」(155 ページ) を参照してください。)

identd の詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] -> [White Papers] にあるホワイトペーパー『Securing Serviceguard』と、identd のマンページを参照してください。

名前解決の構成

Serviceguard は、Linux に組み込まれた名前解決サービスを使用します。

Serviceguard ノードは、クラスターの共有ネットワーク上のどのようなものとも通信できるので、使用しているネットワーク解決サービス (DNS、NIS、または LDAP など) は、対象となるノードのネットワーク上のプライマリアドレスを、プライマリホスト名に解決できなければなりません。

また名前解決は、DNS などのサービスだけに依存するのではなく、各ノードの `/etc/hosts` ファイルにも定義しておくことをお勧めします。ネームサービススイッチは、他のサービスを使う前に `/etc/hosts` ファイルを参照するように構成します。手順については、「[名前解決サービスの障害からの保護](#)」(126 ページ) を参照してください。

注記: クラスター内の通信にプライベート IP アドレスを使用し、これらのアドレスを DNS(または使用する名前解決サービス) が認識していない場合、アドレスを `/etc/hosts` に記述する必要があります。

IPv6 のみのクラスターおよび IPv4 と IPv6 が混在するクラスターに適用される要件と制限事項については、それぞれ、「[IPv6 専用モードの規則と制限事項](#)」(80 ページ) および「[混合モードの規則と制限事項](#)」(82 ページ) を参照してください。また、最新バージョンの Serviceguard リリースノートも参照してください。

たとえば、2 つのプライベートサブネットと 1 つのパブリックサブネットがある 2 ノード (gryf と sly) のクラスターがあるとします。プライベートサブネットを共有していない非クラスターノード (bit) にこれらのノードへのアクセスを許可するとします。両方のクラスターノード上の `/etc/hosts` ファイルには、以下の内容が含まれていなければなりません。

```
15.145.162.131 gryf.uksr.hp.com gryf
10.8.0.131 gryf.uksr.hp.com gryf
10.8.1.131 gryf.uksr.hp.com gryf

15.145.162.132 sly.uksr.hp.com sly
10.8.0.132 sly.uksr.hp.com sly
10.8.1.132 sly.uksr.hp.com sly

15.145.162.150 bit.uksr.hp.com bit
```

Serviceguard ノードの `/etc/hosts` でエントリーを作成する際は、以下の規則を念頭に置いてください。

1. クラスター構成ファイルの `NODE_NAME` は、完全修飾ドメイン名 (ピリオドで区切られた 4 要素から成る名前) の最初の要素に当たるホスト名と一致する必要があります。このホスト名は、`hostname()` コマンドで返される名前です。たとえば、`NODE_NAME` は、`gryf.uksr.hp.com` ではなく `gryf` でなければなりません。詳細は、「[クラスター構成のパラメーター](#)」にある `NODE_NAME` エントリーを参照してください。

注記: Serviceguard が認識するのはホスト名だけなので、`gryf.uksr.hp.com` と `gryf.cup.hp.com` は同じクラスター内のノードとしては存在できません。つまり、Serviceguard では、この 2 つは `gryf` という同一のホストとして扱われます。

2. すべてのプライマリ IP アドレスを構成します。

注記: Serviceguard は、完全修飾ドメイン名 (前述の例に示したような名前) のホスト名 (最初の要素) のみを認識します。たとえば、`gryf.uksr.hp.com` と `gryf.cup.hp.com` は、同じホストの `gryf` とみなされるため、同じクラスター内のノードとして存在できないことになります。

アプリケーションで、ホスト名の別名を使う必要がある場合は、Serviceguard ホスト名は、そのホストの全エントリーにおける別名の 1 つである必要があります。たとえば、前述の例の 2 ノードクラスターを別名のホスト名 `alias-node1` と `alias-node2` を使用するように構成した場合、`/etc/hosts` のエントリーは以下のように記述されます。

```
15.145.162.131 gryf.uksr.hp.com gryf1 alias-node1
10.8.0.131 gryf2.uksr.hp.com gryf2 alias-node1
10.8.1.131 gryf3.uksr.hp.com gryf3 alias-node1
15.145.162.132 sly.uksr.hp.com sly1 alias-node2
10.8.0.132 sly2.uksr.hp.com sly2 alias-node2
10.8.1.132 sly3.uksr.hp.com sly3 alias-node2
```

- ① **重要:** Serviceguard では、IPv6 アドレスのエイリアスはサポートされません。

IPv6 アドレスだけのクラスターやノードのホスト名に IPv6 と IPv4 のアドレスを組み合わせて使用するクラスターの構成については、「[ホスト名アドレスファミリーについて: IPv4 専用、IPv6 専用、および混合モード](#)」(79 ページ) を参照してください。

名前解決サービスの障害からの保護

ユーザーレベルのどの Serviceguard コマンド (cmviewcl など) を使用する場合でも、コマンドは構成されている名前解決サービス (DNS など) を使ってすべてのクラスターノードのアドレスを取得します。名前解決サービスが使用可能でない場合、コマンドはハングするか、予期しないネットワークエラーメッセージを返す可能性があります。

注記: 上記のハングまたはエラーが発生した場合、Serviceguard と保護されているすべてのアプリケーションは、ユーザーが実行したコマンドが動作しない場合でも引き続き動作します。つまり、影響を受けるのは Serviceguard の構成コマンド (および対応する Serviceguard Manager の機能) だけで、クラスターデーモンやパッケージサービスは影響を受けません。

以下の手順では、名前解決サービスが失敗しても、クラスターノード間の通信を続けることができる、堅牢な名前解決構成の作成方法を示します。

1. クラスター内のすべてのノード上の /etc/hosts ファイルを編集します。すべてのハートビート IP アドレスと、すべてのクラスターノードのその他の IP アドレスに対する名前解決を追加します。説明と例については、「[名前解決の構成](#)」(124 ページ) を参照してください。

注記: 各クラスターノードについて、パブリックなネットワークの IP アドレスを最初に記述する必要があります。これにより、他のアプリケーションがパブリックネットワーク上の他のノードと通信できるようになります。

2. DNS を使っている場合は、ネームサーバーが /etc/resolv.conf 内に構成されていることを確認します。例を次に示します。

```
domain cup.hp.com
search cup.hp.com hp.com
nameserver 15.243.128.51
nameserver 15.243.160.51
```

3. すべてのノード上の /etc/nsswitch.conf ファイルを編集または作成し、次の行が存在しなければ追加します。

- DNS の場合は、次の 2 行を追加します。

```
hosts: files [NOTFOUND=continue UNAVAIL=continue] dns [NOTFOUND=return UNAVAIL=return]
ipnodes: files [NOTFOUND=continue UNAVAIL=continue] dns [NOTFOUND=return UNAVAIL=return]
```

- NIS の場合は、次の 2 行を追加します。

```
hosts: files [NOTFOUND=continue UNAVAIL=continue] nis [NOTFOUND=return UNAVAIL=return]
ipnodes: files [NOTFOUND=continue UNAVAIL=continue] nis [NOTFOUND=return UNAVAIL=return]
```

文字列「hosts:」または「ipnodes:」で始まる行がすでに存在する場合は、この文字列のすぐ右側のテキストが、次のようになっていることを確認します (1 行で)。

```
files [NOTFOUND=continue UNAVAIL=continue] dns [NOTFOUND=return UNAVAIL=return]
```

または

```
files [NOTFOUND=continue UNAVAIL=continue] nis [NOTFOUND=return UNAVAIL=return]
```

この手順は、DNS、NIS、プライマリ LAN がダウンした場合でもクラスター内のノードがホスト名を IP アドレスに解決できるようにするためには必須です。

4. クラスターに構成する予定のすべてのノードで `$SGCONF/cmclnodelist` ファイルを作成し、すべてのクラスターノードによるアクセスを許可します。「[未構成ノードへのルートアクセスの許可](#)」(123 ページ) を参照してください。

注記: 複数のネームサーバーを使うか、名前解決サービスを `Serviceguard` パッケージとして構成することで、名前解決サービス自体の可用性も高くすることをお勧めします。

カーネル構成の一貫性

すべてのクラスターノードのカーネル構成は、フェイルオーバー時のクラスターに期待される動作と矛盾していないことを確認します。特にあるクラスターノードでカーネルパラメーターを変更した場合は、同じパッケージを実行する他のクラスターノードでも、同様にパラメーターを変更しておく必要があります。

ネットワークタイムプロトコルの使用

クラスター内の各ノードでは、ネットワークタイムプロトコル (NTP) サービスを使用可能にすることを強くお勧めします。デーモンプロセスとして各システム上で動作する NTP を使用すると、すべてのノードのシステム時刻が等しくなり、整合したタイムスタンプがログファイルに記録され、メッセージサービスの一貫した動作を確保できます。そのため、クラスター内で稼働するアプリケーションの同期が保証されます。NTP サービスデーモン `xntpd` は、すべてのノードで、クラスターの構成を始める前に動かしておく必要があります。NTP 構成ファイルは `/etc/ntp.conf` です。

チャネルボンディングの構成 (Red Hat の場合)

この項の内容は、Red Hat インストールに適用されます。SUSE ディストリビューションを使用している場合には、次の項に進んでください。

LAN インターフェイスのチャネルボンディングは、ボンディングドライバーを使って実現されます。このドライバーはブート時にカーネルに組み込まれます。このドライバーが組み込まれると、ネットワークソフトウェアは `/etc/sysconfig/network-scripts` ディレクトリに作成されている各ボンドのボンディング定義を認識できるようになります。たとえば、`ifcfg-bond0` という名前のファイルは、`bond0` をマスターボンディングユニットとして定義し、`ifcfg-eth0` スクリプトと `ifcfg-eth1` スクリプトは、それぞれのインターフェイスをスレーブとして定義します。

ボンディングはいくつかのモードで定義できます。負荷分散の目的で使用する `mode 0` は、ボンド内のすべてのスレーブデバイスを並列に使用してデータを送信します。これは、LAN インターフェイスカードがイーサネットスイッチに接続されている場合に可能で、スイッチのポートを Fast EtherChannel トランクとして構成する必要があります。パッケージのフェイルオーバーを可能にするには、2 台のスイッチを HA グループとして接続します。

高可用性を実現するには、ボンディングモジュールを `mode 1` で組み込んでください。`mode 1` では、1 つのスレーブがボンドの待機スレーブとして機能し、他のスレーブがデータを送信します。これは、冗長ネットワークハブやスイッチを介した専用ハートビート接続に最適です。

ネットワークボンディングについての詳細は、`kernel-doc rpm` がインストールされていることを確認し、次のファイルを参照してください。

`/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.txt`

注記: ボンディングの構成はシステムコンソールから行うことをお勧めします。これは、構成の終了時にコンソールからネットワークを再起動する必要があるためです。

構成例

LAN の冗長性を確保するために、以下のファイルを構成します。単一のフェイルオーバーに対応するために必要とされるボンドは、1 つだけです。

1. bond0 のファイル ifcfg-bond0 を作成します。

/etc/sysconfig/network-scripts ディレクトリ内に構成ファイルを作成します。たとえば、ファイル ifcfg-bond0 では、bond0 がマスターとして定義されています (実際のシステムでは、192.168.1.1 の部分は、使っているネットワークに応じて適切な値に置き換える必要があります)。

以下の情報を、ifcfg-bond0 ファイルに記述します。

```
DEVICE=bond0
IPADDR=192.168.1.1
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

Red Hat 5 および Red Hat 6 の場合に限り、次の行を ifcfg-bond0 ファイルに追加します。

```
BONDING_OPTS='miimon=100 mode=1'
```

2. ボンド内の各インターフェイスに対して、ifcfg-ethn ファイルを作成します。すべてのインターフェイスに、SLAVE と MASTER を定義する必要があります。たとえば、eth0 と eth1 を使うボンドでは、ifcfg-eth0 ファイルを編集して、以下のようにします。

```
DEVICE=eth0
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
```

ifcfg-eth1 ファイルは、以下のようにします。

```
DEVICE=eth1
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
```

Red Hat 5 および Red Hat 6 の場合に限り、インターフェイスのハードウェア (MAC) アドレスを含む行を ifcfg-ethn スレーブファイルに追加します。次に例を示します。

```
HWADDR=00:12:79:43:5b:f4
```

3. 以下の行を、/etc/modprobe.conf に追加します。

```
alias bond0 bonding options bond0 miimon=100 mode=1
```

2 つ目のボンディングインターフェイスを構成した場合は、bond1 に対して MASTER=bond1 とし、最初のボンド (bond0) の後に次のような行を追加します。

```
options bond1 -o bonding1 miimon=100 mode=1
```

注記: 構成を行うときには、各ノード上で、同じボンドに対するアクティブスレーブが同じハブやスイッチに接続されていることを確認してください。これは各ノード上で /proc/net/bonding/bond<x>/info ファイルを調べることで確認できます。このファイルでは、bond<x> に対するアクティブスレーブを示しています。

ネットワークの再起動

ネットワークサブシステムを再起動します。Red Hat システムの場合は、クラスター内のいずれかのノードの **コンソール** から、次のコマンドを実行します。

```
/etc/rc.d/init.d/network restart
```

注記: クラスターサブネットの外部からネットワークを再起動しないようにしてください。これは、コマンドが完了する前にネットワークが切断される可能性があるためです。

このコマンドを実行すると、「bringing up network」というメッセージが表示されます。いずれかのボンディング構成ファイルにエラーがあった場合、ネットワークは正常に動作しません。その場合は、各構成ファイルの誤りを調べ、ネットワークの再起動を試みてください。

構成の表示

構成と送信ポリシーは、`ifconfig` で確認することができます。上記で作成した構成の場合、次のような表示になります。

`/sbin/ifconfig`

```
bond0      Link encap:Ethernet  HWaddr 00:C0:F0:1F:37:B4
inet addr:192.168.1.1 Bcast:192.168.1.255  Mask:255.255.255.0
UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
RX packets:7224794 errors:0 dropped:0 overruns:0 frame:0
TX packets:3286647 errors:1 dropped:0 overruns:1 carrier:0
collisions:0 txqueuelen:0

eth0       Link encap:Ethernet  HWaddr 00:C0:F0:1F:37:B4
inet addr:192.168.1.1 Bcast:192.168.1.255  Mask:255.255.255.0
UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
RX packets:3573025 errors:0 dropped:0 overruns:0 frame:0
TX packets:1643167 errors:1 dropped:0 overruns:1 carrier:0
collisions:0 txqueuelen:100
Interrupt:10 Base address:0x1080

eth1       Link encap:Ethernet  HWaddr 00:C0:F0:1F:37:B4
inet addr:192.168.1.1 Bcast:192.168.1.255  Mask:255.255.255.0
UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
RX packets:3651769 errors:0 dropped:0 overruns:0 frame:0
TX packets:1643480 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
Interrupt:9  Base address:0x1400
```

チャネルボンディングの構成 (SUSE の場合)

Red Hat ディストリビューションを使用している場合には、1 つ前の項で説明されている手順に従ってください。以下の記述は、SUSE にのみあてはまります。

まず `yast/yast2` を実行し、イーサネットデバイスを DHCP として構成して、`ifcfg-eth-id-<mac>` ファイルを作成します。

次にボンディングしたい各 `ifcfg-eth-id-<mac>` ファイルを変更します。ファイルは `/etc/sysconfig/network` にあります。以下の内容から

```
BOOTPROTO='dhcp'
MTU=''
REMOTE_IPADDR=''
STARTMODE='onboot'
UNIQUE='gZD2.ZqnB7JKTdX0'
_nm_name='bus-pci-0000:00:0b.0'
```

以下の内容に変更します。

```
BOOTPROTO='none'
STARTMODE='onboot'
UNIQUE='gZD2.ZqnB7JKTdX0'
_nm_name='bus-pci-0000:00:0b.0'
```

注記: `UNIQUE` パラメーターと `_nm_name` パラメーターは変更しないでください。`MTU` と `REMOTE_IPADDR` は、設定されていなければそのまま構いません。

次に、`/etc/sysconfig/network` にある `ifcfg-bond0` ファイルを編集して以下のような内容にします。

```
BROADCAST='172.16.0.255'
BOOTPROTO='static'
IPADDR='172.16.0.1'
MTU=''
NETMASK='255.255.255.0'
NETWORK='172.16.0.0'
REMOTE_IPADDR=''
STARTMODE='onboot'
BONDING_MASTER='yes'
BONDING_MODULE_OPTS='miimon=100 mode=1'
BONDING_SLAVE0='eth0'
BONDING_SLAVE1='eth1'
```

この例では、`mii monitor` が 100、モードが `active-backup` の `bond0` を構成しています。個々の構成に合わせて、`IP`、`BROADCAST`、`NETMASK`、および `NETWORK` の各パラメーターを調整してください。

この例でわかるように、構成オプション `BONDING_MASTER`、`BONDING-MODULE_OPTS`、および `BONDING_SLAVE` を追加します。`BONDING-MODULE_OPTS` は、ボンディングモジュールに渡す追加オプションです。`max_bonds` をオプションとして渡すことはできません。必要な各ボンドに対して `ifup` スクリプトがモジュールをロードするため、これは不要です。

`BONDING_SLAVE` は、どのイーサネットデバイスを `bond0` のスレーブにするかを `ifup` に指示します。したがって、4 つのイーサネットデバイスをボンディングしたい場合は、次の行を追加します。

```
BONDING_SLAVE2='eth2'
BONDING_SLAVE3='eth3'
```

注記: `eth` ID と `MAC` アドレスの対応を確認するには、`ifconfig` を使います。

ボンディングに関するネットワーク情報の詳細

は、`/usr/src/linux<kernel_version>/Documentation/networking/bonding.txt` を参照してください。

ネットワークの再起動

ネットワークサブシステムを再起動します。SUSE システムの場合は、クラスター内の任意のノードのコンソールから、次のコマンドを実行します。

```
/etc/init.d/network restart
```

注記: クラスターサブネットの外部からネットワークを再起動しないようにしてください。これは、コマンドが完了する前にネットワークが切断される可能性があるためです。

いずれかのボンディング構成ファイルにエラーがある場合、ネットワークが正常に動作しないことがあります。その場合は、各構成ファイルの誤りを調べ、ネットワークの起動を再度試みてください。

ロック LUN の設定

ロック LUN は、`fdisk` コマンドでタイプ `linux (83)` として定義された、少なくとも 100K の 1 シリンダーのパーティションを必要とします。各クラスターノードで表示される、ロック LUN のパス名が必要となります。単一のノードで `fdisk` コマンドを使用して、この LUN をシリンダー 1、タイプ 83 のパーティションとして定義します。次に設定例を示します。

プロンプトに対して下記の表に示すとおりに応答し、ロック LUN パーティションを設定します。

```
fdisk <Lock LUN Device File>
```

表 7 Linux パーティションタイプの変更

	プロンプト	応答	実行される動作
1.	Command (m for help):	n	新しいパーティションの作成
2.	Partition number (1-4):	1	影響のあるパーティション
3.	Hex code (L to list codes):	83	パーティションのタイプをデフォルトである Linux に設定
4.	Command (m for help):	1	最初のパーティションを設定
5.	Command (m for help):	1	サイズを 1 シリンダーに設定
6.	Command (m for help):	p	パーティションデータを表示する
7.	Command (m for help):	w	パーティションテーブルにデータを書き込む

以下に示す fdisk の対話例は、デバイスファイル /dev/sdc が Smart Array タイプのパーティションに設定されることを示しています。以下のように表示されます。

```
fdisk /dev/sdc
```

```
Command (m for help): n
Partition number (1-4): 1
HEX code (type L to list codes): 83
Command (m for help): 1
Command (m for help): 1
```

```
Command (m for help): p
Disk /dev/sdc: 64 heads, 32 sectors, 4067 cylinders
Units = cylinders of 2048 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdc		1	1	1008	83	Linux

```
Command (m for help): w
The partition table has been altered!
```

注記: 以下の規則に従ってください。

- LVM を使ってロック LUN を構成しないでください。
- パーティションタイプは 83 でなくてはなりません。
- ロック LUN 用のパーティションには、ファイルシステムを作成しないでください。
- ロック LUN に対して、md を使ってマルチパスを構成しないでください。

ディスクパーティションのフォーマットをクラスター内の他のノードに転送するには、以下のコマンドを使用します。

```
sfdisk -R <device>
```

ここで、<device>は最初のノード上の物理デバイスです。たとえば、もう一方のノード上でのデバイス名が /dev/sdc であれば、下記のコマンドを実行します。

```
sfdisk -R /dev/sdc
```

パーティションテーブルを確認するには、下記のコマンドを実行します。

```
fdisk -l /dev/sdc
```

注記: SUSE では、fdisk が使えないプラットフォームがあります。その場合は、YAST2 を使ってパーティションを設定して構いません。

クォーラムサーバーの設定と実行

ロック LUN でなく、クォーラムサーバーを使用する場合、Quorum Server ソフトウェアは、クラスターが稼働するノード以外のシステム上にインストールしなければなりません。また、クラスターの構成時に稼働している必要があります。

クォーラムサーバーのインストール、アップデート、構成、および実行についての説明、推奨事項、および手順については、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にある『HP Serviceguard Quorum Server Version A.04.00 Release Notes』を参照してください。「[クラスター構成のパラメーター](#)」(82 ページ)にあるパラメーター `QS_HOST` および `QS_ADDR` の説明も参照してください。

論理ボリュームインフラストラクチャの作成

Serviceguard は、共有ディスクストレージを使います。この領域は、冗長データストレージと共有デバイスへの冗長パスを使って、高可用性が確保されます。Serviceguard パッケージのストレージは、ノード上のパッケージの開始の一環としてノード上でアクティブにされる LVM ボリュームグループから論理的に構築されます。ストレージは、一般的に、論理ユニット (LUN) 上に構成されます。

Serviceguard パッケージのためのディスクストレージは、複数のクラスターノードにケーブル接続された共有ディスクに構築されます。これらのディスクストレージは、ブートパーティションとルートファイルシステムを持つプライベートな Linux ルートディスクとは別のものです。共有ディスクにアプリケーションのためのデータスペースを用意するためには、fdisk ユーティリティを使ってディスクパーティションを作成し、LVM によって論理ボリュームを構築します。

共有データストレージにボリュームグループを定義する前または後に、クラスターを構築できます (後述の項を参照)。最初にクラスターを作成する場合は、ボリュームグループを作成した後に、ストレージ情報をクラスター構成ファイルとパッケージ構成ファイルに追加します。

HP Serviceguard for Linux でのボリューム管理の概要については、「[データストレージ用のボリュームマネージャー](#)」(64 ページ)を参照してください。ここでは、以下の作業を行う方法について説明します。

- [ディスク情報の表示](#) (133 ページ)
- [パーティションの作成](#) (133 ページ)
- [ボリュームグループのアクティブ化保護を有効にする](#) (135 ページ)
- [ボリュームグループの構築: Smart Array クラスタストレージ \(MSA 2000 シリーズ\) の例](#) (136 ページ)
- [ボリュームグループと論理ボリュームの構築](#) (137 ページ)
- [すべてのノードへの共有構成の配布](#) (138 ページ)
- [共有構成のテスト](#) (138 ページ)

- ボリュームグループ構成データの保存 (140 ページ)
- ディスクモニターのセットアップ (141 ページ)

△ 注意: LVM ボリュームグループのマイナー番号は、すべてのクラスターノード上で同じでなければなりません。したがって、クラスター内に非共有のボリュームグループがある場合は、すべてのノード上で、非共有のボリュームグループを同じ数だけ、共有ストレージの定義の**前**に作成してください。できる限り、プライベートボリュームグループ (特に LVM ブートボリューム) の使用は避けてください。各論理ボリュームでマイナー番号がインクリメントされ、論理ボリュームの数がノード間で一致しないと、LVM の障害を引き起こす可能性があります (LVM ブートボリュームを使用している場合は、ブートされます)。

注記: 後続のセクションで言及されている場合を除いて、共有ストレージの LVM の構成は、1 つのノード上でのみ行います。他のノードをリブートすると、ディスクパーティションがこれらのノードから参照できるようになります。LVM 構成をすべてのクラスターノードに配布すると、LVM コマンドを使って、ノード間でボリュームグループを切り替えることができるようになります。(データ破壊を防ぐために、各ボリュームグループは、同時に複数のノードでアクティブにすることはできません。)

マルチパスについては、「[ストレージのマルチパス](#)」(75 ページ) を参照してください。

ディスク情報の表示

構成済みのディスクのリストを表示するには、次のコマンドを使います。

```
fdisk -l
```

次のように表示されます。

```
Disk /dev/sda: 64 heads, 32 sectors, 8678 cylinders
Units = cylinders of 2048 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1	*	1	1001	1025008	83	Linux
/dev/sda2		1002	8678	7861248	5	Extended
/dev/sda5		1002	4002	3073008	83	Linux
/dev/sda6		4003	5003	1025008	82	Linux swap
/dev/sda7		5004	8678	3763184	83	Linux

```
Disk /dev/sdb: 64 heads, 32 sectors, 8678 cylinders
Units = cylinders of 2048 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
--------	------	-------	-----	--------	----	--------

```
Disk /dev/sdc: 255 heads, 63 sectors, 1106 cylinders
```

```
Units = cylinders of 16065 * 512 bytesDisk /dev/sdd: 255 heads, 63 sectors, 1106 cylinders
Units = cylinders of 16065 * 512 bytes
```

この例では、デバイスファイル /dev/sda で示されたディスクはすでに Linux 用にパーティションが設定されています (/dev/sda1 ~ /dev/sda7 というパーティション)。2 番目の内部デバイス /dev/sdb と 2 つの外部デバイス /dev/sdc および /dev/sdd には、パーティションが設定されていません。

注記: SUSE では、fdisk が使えないプラットフォームがあります。その場合は、YAST2 を使ってパーティションを設定して構いません。

パーティションの作成

共有ストレージ用として使う各ディスクデバイス (個別のディスクや、アレイ内の LUN) 上に、パーティションを定義しなければなりません。パーティションの定義には、fdisk コマンドを使います。

以下の手順で、新しいパーティションを作成します。

1. fdisk を実行します。<DeviceName> には、デバイスファイル名を指定します。

```
# fdisk <DeviceName>
```

パーティションを定義するには、プロンプトに対して、次の表のように応答します。

	プロンプト	応答	実行される動作
1.	Command (m for help):	n	新しいパーティションを作成する
2.	Command action e extended p primary partition (1-4)	p	プライマリパーティションを作成する
3.	Partition number (1-4):	1	パーティション 1 を作成する
4.	First cylinder (1- <i>nn</i> , default 1):	以下を入力します。	開始シリンダー 1 (デフォルト) を指定する
5.	Last cylinder or +size or +sizeM or +sizeK (1- <i>nn</i> , default <i>nn</i>):	以下を入力します。	最後のシリンダー番号 (デフォルト) を指定する
6.	Command (m for help):	p	パーティションデータを表示する
7.	Command (m for help):	w	パーティションテーブルにデータを書き込む

次の fdisk の例では、デバイスファイル /dev/sdc 上のディスク全体を 1 つのパーティションとして構成しています。次のように表示されます。

```
fdisk /dev/sdc
```

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4) p
Partition number (1-4): 1
First cylinder (1-4067, default 1): Enter
Using default value 1Last cylinder or +size or +sizeM or +sizeK (1-4067, default 4067): Enter
Using default value 4067
```

```
Command (m for help): p
Disk /dev/sdc: 64 heads, 32 sectors, 4067 cylinders
Units = cylinders of 2048 * 512 bytes
```

```
Device Boot      Start         End      Blocks   Id  System
/dev/sdc                1         4067     4164592    83  Linux
```

```
Command (m for help): w
The partition table has been altered!
```

2. パーティションのタイプを設定するには、各プロンプトに対して次の表のように応答します。

	プロンプト	応答	実行される動作
1.	Command (m for help):	t	パーティションタイプを設定する
2.	Partition number (1-4):	1	影響のあるパーティション
3.	Hex code (L to list codes):	8e	パーティションタイプを Linux LVM に設定する
4.	Command (m for help):	p	パーティションデータを表示する
5.	Command (m for help):	w	パーティションテーブルにデータを書き込む

次の fdisk の例では、デバイスファイル /dev/sdc 上のディスクを Smart Array タイプのパーティションに設定しています。次のように表示されます。

```
fdisk /dev/sdc
```

```
Command (m for help): t
```

```
Partition number (1-4): 1
```

```
HEX code (type L to list codes): 8e
```

```
Command (m for help): p
```

```
Disk /dev/sdc: 64 heads, 32 sectors, 4067 cylinders
```

```
Units = cylinders of 2048 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdc		1	4067	4164592	8e	Linux LVM

```
Command (m for help): w
```

```
The partition table has been altered!
```

3. この手順を、共有ストレージとして使う各デバイスファイルに対して行います。

```
fdisk /dev/sdd
```

```
fdisk /dev/sdf
```

```
fdisk /dev/sdg
```

4. 内蔵ストレージ用のボリュームグループを作成する場合、それらのパーティションも作成し、共有ストレージの定義の前にそのボリュームグループを作成してください。

```
fdisk /dev/sddb
```

注記: SUSE では、fdisk が使えないプラットフォームがあります。その場合は、YAST2 を使ってパーティションを設定して構いません。

ボリュームグループのアクティブ化保護を有効にする

Serviceguard for Linux A.11.16.07 以降では、ボリュームグループのアクティブ化保護機能を有効にして、ボリュームグループが一度に複数のノードでアクティブ化されるのを防止することができます。アクティブ化保護機能を使用する場合は、クラスターノードごとに有効にする必要があります。

Red Hat および SUSE のシステムでボリュームグループのアクティブ化保護を有効にするには、以下の手順を実行します。

- ① **重要:** それぞれのノードでこの手順を実行してください。

1. /etc/lvm/lvm.conf を編集し、次の行を追加します。

```
tags { hosttags = 1 }
```

2. /etc/lvm/lvm.conf にある、# volume_list = から始まる行をコメントアウトし、ルートボリュームグループなど、ノードの「プライベート」ボリュームグループ (他のクラスターノードと共有されていないボリュームグループ) をすべて含めるようにこれを編集します。

たとえば、ルートボリュームグループが vg00 で、ノードが vg01 と vg02 もプライベートボリュームグループとして使用する場合は、この行を次のように編集します。

```
volume_list = [ "vg00", "vg01", "vg02" ]
```

3. ファイル `/etc/lvm/lvm_$(uname -n).conf` を作成します。
4. 手順 2 で作成したファイルに、次の行を追加します。

```
activation { volume_list=["@node"] }
```

`node` は、`uname -n` の値です。

5. `vgscan` を実行します。

```
vgscan
```

注記: これでボリュームグループのアクティブ化保護の設定は完了です。Serviceguard では、パッケージが起動される際に、所有しているノードの `uname -n` の値に一致するタグがパッケージに定義された各ボリュームグループに追加されます。このタグは、パッケージが停止されたときに削除されます。コマンド `vgc -o +tags vgname` を実行すると、ボリュームグループに設定されているタグがすべて表示されます。

以降の項では、ボリュームグループおよび論理ボリュームの設定と共有構成の配布のプロセスについて説明します。このプロセスを終了したら、「[共有構成のテスト](#)」(138 ページ)の手順に従い、設定が正しく行われたことを確認します。

ボリュームグループの構築: Smart Array クラスタストレージ (MSA 2000 シリーズ) の例

注記: Serviceguard で使用する MSA 2000 の設定および構成についての詳細は、<http://www.hp.com/go/linux-serviceguard-docs> にある『HP Serviceguard for Linux Version A.11.19 or later Deployment Guide』を参照してください。

システムの Logical Volume Manager (LVM) を使って、Serviceguard パッケージがアクティブ化するボリュームグループを作成します。ここでは、MSA 2000 シリーズのストレージ上に作成した LUN 上にボリュームグループを作成する例を示します。LVM についての詳細は、<http://www.tldp.org/HOWTO/HOWTO-INDEX/howtos.html> にある『Logical Volume Manager HOWTO』を参照してください。

開始する前に、パーティションタイプ 8e (Linux LVM) を使って LUN にパーティションを作成し、それらに名前を付けておいてください。デフォルトの 83 (Linux) を変更するには、`fdisk` コマンドのタイプ `t` パラメーターを使用します。

1 台のノード上で以下の手順を実行します。

1. LVM 構成をアップデートし、`/etc/lvmtab` ファイルを作成します。そのノード上でボリュームグループが作成済みの場合は、このステップを省略できます。

```
vgscan
```

注記: ディストリビューションによっては、ファイル `/etc/lvmtab` と `/etc/lvmtab.d` が存在しない場合があります。その場合は、これらのファイルについての記述を無視してください。

2. 各 LUN 上に LVM 物理ボリュームを作成します。例:

```
pvccreate -f /dev/sda1
```

```
pvccreate -f /dev/sdb1
```

```
pvccreate -f /dev/sdc1
```

3. このノード上にすでに定義されているボリュームグループがあるかどうかをチェックします。各ボリュームグループには重複しない名前を付けるようにしてください。

```
vgdisplay
```

4. 定義する各 Serviceguard パッケージに対して、個別のボリュームグループを作成します。以下の例では、LUN /dev/sda1 と /dev/sdb1 をボリュームグループ vgpkgA に、/dev/sdc1 を vgpkgB に追加します。

```
vgcreate --addtag $(uname -n) /dev/vgpkgA /dev/sda1 /dev/sdb1
```

```
vgcreate --addtag $(uname -n) /dev/vgpkgB /dev/sdc1
```

注記: vgchange --addtag は、ボリュームグループのアクティブ化保護を実装する場合にのみ使用します。ボリュームグループのアクティブ化保護を使用する際は、**それぞれ**のノードで有効にしなければなりません。

ボリュームグループと論理ボリュームの構築

1. Logical Volume Manager (LVM) を使って、Serviceguard パッケージがアクティブ化するボリュームグループを作成します。
LUN 上にボリュームグループを作成する方法を示す例については、「[ボリュームグループの構築: Smart Array クラスタストレージ \(MSA 2000 シリーズ\) の例](#)」(136 ページ)を参照してください。(ファイバーチャネルストレージの場合は、「[パーティションの作成](#)」(133 ページ) で使ったデバイスファイル名を使います。)
2. ボリュームグループのアクティブ化保護機能をサポートしている Linux ディストリビューションでは、この機能を有効にします。「[ボリュームグループのアクティブ化保護を有効にする](#)」(135 ページ)を参照してください。
3. これらのボリュームグループ上にデータを格納するには、論理ボリュームを作成しなければなりません。次のコマンドは、/dev/vgpkgA/lvol1 という名前の 500 MB の論理ボリュームと、/dev/vgpkgA/lvol2 という名前の 1 GB の論理ボリュームを、vgpkgA ボリュームグループ内に作成します。

```
lvcreate -L 500M vgpkgA
```

```
lvcreate -L 1G vgpkgA
```

4. これらの論理ボリュームのいずれかにファイルシステムを作成し、新しく作成したディレクトリにこのファイルシステムをマウントします。

```
mke2fs -j /dev/vgpkgA/lvol1
```

```
mkdir /extra
```

```
mount -t ext3 /dev/vgpkgA/lvol1 /extra
```

注記: サポートされているファイルシステムタイプについては、[\(176 ページ\)](#)の `fs_type` の説明を参照してください。

5. /extra ファイルシステムが正しく作成され、高可用性が確保されていることをテストするには、ファイルシステム上にファイルを作成し、読み取りを行います。

```
echo "Test of LVM" >> /extra/LVM-test.conf
```

```
cat /extra/LVM-test.conf
```

注記: YAST や YAST2 を使ってボリュームグループを構成すると、システム上の**すべての**ボリュームグループがアクティブ化されることがあるので注意してください。YAST または YAST2 を実行した後は、現在実行中でない Serviceguard パッケージ用のボリュームグループがアクティブ化されていないことを確認し、アクティブ化されている場合は、LVM コマンドを使って非アクティブ化します。たとえば、ボリュームグループ `sgvg00` を非アクティブ化するには、コマンド `vgchange -a n /dev/sgvg00` を実行します。

すべてのノードへの共有構成の配布

論理ボリュームインフラストラクチャの設定のゴールは、クラスター内の複数のノード上でアクティブ化できるボリュームグループセットの構築にあります。このためには、同じパッケージを実行するすべてのノード上に同じ LVM ボリュームグループを構築する必要があります。

注記: LVM ボリュームグループのマイナー番号は、すべてのクラスターノード上で同じでなければなりません。すべてのノードに同じ数の非共有ボリュームグループがあれば、このようになります。

共有構成を配布するには、以下の手順を実行します。

1. ボリュームグループをアンマウントして非アクティブ化し、必要に応じてタグを削除します。たとえば、`vgpkgA` だけを非アクティブ化するには、次のコマンドを実行します。

```
umount /extra
```

```
vgchange -a n vgpkgA
```

```
vgchange --deltag $(uname -n) vgpkgA
```

注記: `vgchange --deltag` は、ボリュームグループのアクティブ化保護を実装している場合にのみ使用します。ボリュームグループのアクティブ化保護を使用する際は、**それぞれの**ノードで有効にしなければなりません。

2. ノード `ftsys9` 上で作成した新しいディスクパーティションをノード `ftsys10` から認識できるようにするには、リブートします。

リブート

リブートされたノード上のパーティションテーブルは、ディスクに保存されている、他のノードで作成されたパーティション情報を使って再構築されます。

注記: ここで**必ず**リブートしてください。

3. `vgscan` を実行して新しいノード上で LVM 構成を参照できるようにし、`/etc/lvmtab` と `/etc/lvmtab.d` 上に LVM データベースを作成します。たとえば、`ftsys10` 上で次のコマンドを実行します。

```
vgscan
```

共有構成のテスト

共有ボリュームグループの構成が完了したら、次の手順で、ストレージが正しく共有されているかテストします。

1. `ftsys9` で、ボリュームグループをアクティブ化し、その上に構築されたファイルシステムをマウントし、共有ファイルシステム内にファイルを書き込み、結果を確認します。

```
vgchange --addtag $(uname -n) vgpkgB
```

注記: Serviceguard for Linux のボリュームグループのアクティブ化保護機能を使用している場合、ボリュームグループを手動でアクティブ化するときに、`vgchange --addtag` を使用してタグを追加する必要があります。同様に、パッケージで使用するボリュームグループを非アクティブ化する際には、各手順の最後に示されているように、タグを削除する必要があります。

`vgchange --addtag` と `vgchange --deltag` は、ボリュームグループのアクティブ化保護を実装している場合にのみ使用します。ボリュームグループのアクティブ化保護を使用する際は、**それぞれの**ノードで有効にしなければなりません。

Serviceguard では、パッケージが起動される際に、所有しているノードの `uname -n` の値に一致するタグが、パッケージに定義された各ボリュームグループに追加されます。このタグは、パッケージが停止されたときに削除されます。コマンド `vgs -o +tags vgvname` を実行すると、ボリュームグループに設定されているタグがすべて表示されます。

```
vgchange -a y vgpkgB
mount /dev/vgpkgB/lvol1 /extra
echo 'Written by' `hostname` 'on' `date` > /extra/datestamp
cat /extra/datestamp
```

次のように、他のノードによって書き込まれた日付と時刻が表示されます。

```
Written by ftsys9.mydomain on Mon Jan 22 14:23:44 PST 2006
```

次に、再度ボリュームグループをアンマウントします。

```
umount /extra
vgchange -a n vgpkgB
vgchange --deltag $(uname -n) vgpkgB
```

2. ftsys10 上で、ボリュームグループをアクティブ化し、ファイルシステムをマウントし、共有ファイルに日付と時刻を書き込んで、ファイルの内容を確認します。

```
vgchange --addtag $(uname -n) vgpkgB
vgchange -a y vgpkgB
mount /dev/vgpkgB/lvol1 /extra
echo 'Written by' `hostname` 'on' `date` >> /extra/datestamp
cat /extra/datestamp
```

次のように表示されます。他のノードによって書き込まれた日付と時刻が含まれています。

```
Written by ftsys9.mydomain on Mon Jan 22 14:23:44 PST 2006
```

```
Written by ftsys10.mydomain on Mon Jan 22 14:25:27 PST 2006
```

ここで再度ボリュームグループをアンマウントし、手順 1 で追加したタグを削除します。

```
umount /extra
vgchange -a n vgpkgB
vgchange --deltag $(uname -n) vgpkgB
```

注記: Serviceguard for Linux のボリュームグループのアクティブ化保護機能では、ボリュームグループを手動でアクティブ化する際に、上記手順の最初に示したように、タグを追加する必要があります。同様に、パッケージで使用するボリュームグループを非アクティブ化する際には、各手順の最後に示されているように、タグを削除する必要があります。Serviceguard for Linux A.11.16.07 からは、パッケージが起動される際に、所有しているノードの `uname -n` の値に一致するタグが、パッケージに定義されている各ボリュームグループに自動的に追加されるようになりました。このタグは、パッケージが停止される際に削除されます。コマンド `vgs -o +tags vgname` を実行すると、ボリュームグループに設定されているタグがすべて表示されます。

ボリュームグループ構成データの保存

ボリュームグループを作成したときに、LVM はボリュームグループ構成のバックアップコピーを構成ノード上に作成します。さらに、`vgcfgbackup` コマンドを使って、そのボリュームグループがアクティブ化されるすべてのノード上の構成データのバックアップを作成する必要があります。

```
vgcfgbackup vgpkgA vgpkgB
```

ボリュームグループ内のディスクを交換しなければならない場合は、`vgcfgrestore` コマンドを使って古いディスクのメタデータを新しいディスクに復元できます。第 8 章「クラスタのトラブルシューティング」の「ディスクの交換」を参照してください。

ブート時に `vgscan` が実行されないようにし、Serviceguard ボリュームグループがアクティブにならないようにする

デフォルトでは、システムをリブートするたびに Linux は LVM の起動動作を実行します。これには、`vgscan` (いくつかの Linux ディストリビューション) とボリュームグループのアクティブ化が含まれます。これにより、Serviceguard 環境で使われるボリュームで問題が発生する可能性があります (たとえば、現在実行中でない Serviceguard パッケージ用のボリュームグループがアクティブ化されることがあります)。この問題を避けるには、各 Linux バージョンで以下の手順を実行します。

注記: 「ボリュームグループのアクティブ化保護を有効にする」(135 ページ) で説明したとおりにボリュームグループのアクティブ化保護を実装した場合は、これらの手順を実行する必要はありません。

SUSE Linux Enterprise Server

ブート時に `vgscan` が実行されないようにするには、すべてのクラスタードから `/etc/rc.d/boot.d/S07boot.lvm` ファイルを削除します。

注記: YAST または YAST2 を使ってボリュームグループを構成すると、**すべての**ボリュームグループがアクティブ化される可能性があるため、注意してください。YAST または YAST2 を実行した後で、現在実行中でない Serviceguard パッケージのボリュームグループがアクティブ化されていないことを確認し、アクティブ化されている場合には、LVM コマンドを使って非アクティブ化してください。たとえば、ボリュームグループ `sgvg00` を非アクティブ化するには、コマンド `vgchange -a n /dev/sgvg00` を実行します。

Red Hat

Red Hat では、`vgscan` が実行されないようにする必要はありません。

Serviceguard の制御下にあるボリュームグループを非アクティブ化するには、`vgchange` コマンドを `/etc/rc.d/rc.sysinit` の最後に追加します。たとえば、ボリュームグループ `sgvg00` と `sgvg01` が Serviceguard の制御下にある場合は、ファイルの最後に次の行を追加します。

```
vgchange -a n /dev/sgvg00
```

```
vgchange -a n /dev/sgvg01
```

vgchange コマンドは、一時的にボリュームグループをアクティブ化してから非アクティブ化します。これは正常な動作です。

ディスクモニターのセットアップ

HP Serviceguard for Linux にはディスクモニターが含まれています。このディスクモニターを使用して、ディスク接続の障害を検出できます。ディスクのリンク障害が発生すると、ディスクモニターはパッケージを別のノードへフェイルオーバーします。

ディスクモニターの構成方法については、「[ディスク監視構成の作成](#)」(184 ページ)を参照してください。

クラスターの構成

ここでは、基本的なクラスター構成を定義する方法について説明します。この作業は、Serviceguard クラスターに参加していないシステム (すなわち、Serviceguard はインストールされているものの、まだ構成されていないシステム) で行う必要があります。この作業は、Serviceguard Manager で行うことも、下記のようにコマンド行から行うこともできます。

cmquerycl コマンドを使用して、クラスター内のノードセットを指定し、クラスター構成ファイルのテンプレートを生成します。

- ❶ **重要:** 「[クラスター構成のパラメーター](#)」(82 ページ)にある `NODE_NAME` で、ノード名の制限についての重要な情報を参照してください。

コマンドの例を示します (全体を 1 行で入力します)。

```
cmquerycl -v -C $SGCONF/clust1.conf -n ftsys9 -n ftsys10
```

このコマンドにより、テンプレートファイル(デフォルトでは `/etc/cmcluster/clust1.conf`)が作成されます。この出力ファイルでは、キーワードと定義はスペースによって区切られています。コメントを記述することもでき、コメントの一番左のカラムには番号記号(#)を付けて、コメントであることを示します。

注記: あらゆるネットワークでハートビートを送信できるように、ファイルを修正することを強くお勧めします。

cmquerycl コマンドのマニュアルページでは、このファイルに出力されるパラメーターを詳しく説明しています。これらの多くについては、[第4章「HA クラスターのプランニングと文書化」](#)(72 ページ)でも説明しています。必要に応じて `/etc/cmcluster/clust1.config` ファイルを修正してください。

cmquerycl のオプション

処理の高速化

クラスターに多数のノード、ネットワーク、またはディスクが接続された大規模な構成または複雑な構成の場合は、cmquerycl コマンドが完了するのに数分かかる場合があります。この構成プロセスにかかる時間を短縮するには、`-k` および `-w` オプションを使用して、選択した情報だけを返すようコマンドに指示できます。

`-k` を指定すると、ディスクブローピングの一部が省略され、クラスターロックのボリュームグループと物理ボリュームに関する情報は返されません。

`-w local` を指定すると、ローカルネットワークのブローピングを指定できます。これにより、ローカルネットワークの各ノード内のインターフェイス間の LAN 接続を確認できます。これは、cmquerycl に `-c` オプションを指定したときのデフォルトの動作です。

(クロスサブネット構成でノードとサブネットを検出する必要がある場合は、`-w local` を使用しないでください。「[全ネットワークのブローピング](#)」を参照してください。)

-w none を指定すると、ネットワークの照会が実行されません。最近、ネットワークのチェックを行ったばかりであれば、このオプションを使うことで時間を節約することができます。

クラスターホスト名のアドレスファミリの指定

-a オプションを使用して、Serviceguard にクラスターノード名 (ならびに、存在する場合、クォーラムサーバーのホスト名) の「IPv4 アドレスのみ」(-a ipv4)、「IPv6 アドレスのみ」(-a ipv6)、または「両方」(-a any) への解決を指示することができます。アドレスファミリは、クラスター構成ファイルの `HOSTNAME_ADDRESS_FAMILY` を使用して構成することもできます。

- ❗ **重要:** IPv6 専用モードおよび混合モードの重要な制限事項を含む詳細な説明については、「[ホスト名アドレスファミリについて: IPv4 専用、IPv6 専用、および混合モード](#)」(79 ページ) を参照してください。

-a オプションを使用すると、Serviceguard は既存のクラスター構成ファイルに `HOSTNAME_ADDRESS_FAMILY` パラメーターの値が存在する場合でもこれを無視して、クラスターおよびクォーラムサーバーのホスト名を -a オプションの指定に従い解決しようとします。

- -a ipv4 を指定すると、各ホスト名は 1 つ以上の IPv4 アドレスに解決しなければならず、できない場合、コマンドは失敗します。
- 同様に、-a ipv6 を指定すると、各ホスト名は 1 つ以上の IPv6 アドレスに解決しなければならず、できない場合、コマンドは失敗します。
- -a any を指定すると、Serviceguard は、各ホスト名の IPv4 アドレスへの解決を試み、それが失敗すると、IPv6 アドレスへの解決を試みます。

-a オプションを使用しない場合は、Serviceguard は次の処理を実行します。

- クラスターが構成済みの場合、Serviceguard は、`HOSTNAME_ADDRESS_FAMILY` で設定されている値 (デフォルトでは IPv4) を使用します。
- クラスターが構成されていない場合、Serviceguard は、ローカルノード (cmquerycl コマンドの実行ノード) のホスト名に対応する IPv4 アドレスを 1 つでも見つけると、すべてのホスト名を IPv4 アドレスに解決しようとします。あるホスト名について対応する IPv4 アドレスが見つからない場合は、IPv6 アドレスを探します。(これは、-a any を指定したのと同じ処理です。)

ハートビートのアドレスファミリの指定

ハートビートに IPv4 アドレスのみまたは IPv6 アドレスのみを使うように Serviceguard に指示するには、-h オプションを使います。たとえば、IPv6 アドレスのみを使う場合、次のように指定します。

```
cmquerycl -v -h ipv6 -C $SGCONF/clust1.conf -n ftsys9 -n ftsys10
```

- -h ipv4 は、IPv4 サブネットのみを検出して構成するよう Serviceguard に指示します。該当するサブネットが見つからない場合、コマンドは失敗します。
- -h ipv6 は、ipv6 サブネットのみを検出して構成するよう Serviceguard に指示します。該当するサブネットが見つからない場合、コマンドは失敗します。
- -h オプションを指定しないと、最小要件を満たすための最適な構成が Serviceguard によって選択されます。IPv4 LAN と IPv6 LAN の両方が利用できる場合は、IPv4 LAN が優先されます。この結果、IPv4 のみ、IPv6 のみ、またはこの両方を混合した構成となります。Serviceguard のデフォルトの選択肢は、「[クラスター構成のパラメーター](#)」(82 ページ) で説明している `HEARTBEAT_IP` パラメーターを使って変更できます。には、ハートビート要件も示されています。
- -h オプションと -c オプションは、相互排他に関係にあります。

クラスターロックの指定

cmquerycl コマンド行を使用して、クラスターロック LUN (-L lock_lun_device) またはクォーラムサーバー (-q quorum_server [qs_ip2]) を指定できます。詳細は、cmquerycl (1m) のマンページを参照してください。

詳細は、「[ロック LUN の指定](#)」と「[クォーラムサーバーの指定](#)」の各項を参照してください。

全ネットワークのブローピング

-w full を指定すると、全ネットワークのブローピングを指定できます。これにより、同じサブネット上にあるかどうかにかかわらず、クラスター内のすべてのノードの LAN インターフェイス間で現在の接続を確認できます。

注記: このオプションは、クロスサブネット構成に構成済みまたは構成可能なノードとサブネットを検出するために使います。「[クロスサブネット情報の取得](#)」(144 ページ)を参照してください。このオプションは、IP Monitor ポーリングターゲットも検証します。詳細は、「[LAN インターフェイスの監視と障害の検出: IP レベル](#)」(59 ページ)と、「[クラスター構成のパラメーター](#)」(82 ページ)にある POLLING_TARGET を参照してください。

ロック LUN の指定

2 ノードのクラスターには、クラスターロック LUN またはクォーラムサーバーが必要です。ロック LUN を使用する場合には、必ず cmquerycl コマンドに -L lock_lun_device オプションを指定してください。デバイスの名前がすべてのノード上で同じ場合には、以下の例に示すように、ノード名の前にオプションを入力します (すべて 1 行で入力)。

```
cmquerycl -v -L /dev/sda1 -n lp01 -n lp02 -C $SGCONF/lpcluster.conf
```

デバイスの名前がノードごとに異なる場合には、以下の例に示すように、各ノード名に続いて各デバイスファイルを指定します (すべて 1 行で入力)。

```
cmquerycl -v -n node1 -L /dev/sda1 -n node2 -L /dev/sda2 -C  
$SGCONF/lpcluster.conf
```

クォーラムサーバーの指定

- ❶ **重要:** 以下に標準的な作業手順を示します。個々の Serviceguard バージョンとクォーラムサーバーに適用される個別の作業手順は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にある最新版の『HP Serviceguard Quorum Server Version A.04.00 Release Notes』の「Configuring Serviceguard to Use the Quorum Server」を参照してください。

2 ノードのクラスターには、クラスターロック LUN またはクォーラムサーバーが必要です。Quorum Server パラメーターを含むクラスター構成ファイルを取得するには、次の例のように cmquerycl コマンドの -q オプションを使って、Quorum Server のホスト名または IP アドレスを指定します (すべて 1 行で入力)。

```
cmquerycl -q <QS_Host> -n ftsys9 -n ftsys10 -C <ClusterName>.conf
```

クォーラムサーバーに到達できる代替ホスト名または IP アドレスを指定するには、次のようなコマンドを使います (すべてを 1 行で入力)。

```
cmquerycl -q <QS_Host> <QS_Addr> -n ftsys9 -n ftsys10 -C  
<ClusterName>.conf
```

QS_HOST (SLES 11 では IPv4 または IPv6、Red Hat 5 および Red Hat 6 では IPv4 のみ)、オプションの QS_ADDR (SLES 11 では IPv4 または IPv6、Red Hat 5 および Red Hat 6 では IPv4 のみ)、QS_POLLING_INTERVAL を入力し、必要に応じて QS_TIMEOUT_EXTENSION を入力します。さらに HOSTNAME_ADDRESS_FAMILY 設定もチェックします (デフォルトは IPv4)。[クラスター構成のパラメーター](#) (82 ページ) にあるパラメーターの説明を参照してください。

重要情報は、「[ホスト名アドレスファミリーについて: IPv4 専用、IPv6 専用、および混合モード](#)」(79 ページ) および「[オンラインでフォーラム構成を変更したときの動作](#)」(37 ページ) も参照してください。

クロスサブネット情報の取得

Serviceguard A.11.18 以降では、ルーターで結合された複数の IPv4 サブネットを、クラスターハートビートとデータの両方で利用するように構成することができます。この構成では、一部のノードがあるサブネットを利用し、別のノードが別のサブネットを利用します。規則および定義については、「[クロスサブネット構成](#)」(25 ページ) を参照してください。

利用可能なサブネットを検出するには、cmquerycl に -w full オプションを指定して実行しなければなりません。

たとえば、NodeA、NodeB、NodeC、および NodeD の 4 ノードを、サブネット 15.13.164.0、15.13.172.0、15.13.165.0、15.13.182.0、15.244.65.0、および 15.244.56.0 を使用するクラスターに構成する予定があるとします。

次のコマンドを入力します。

```
cmquerycl -w full -n nodeA -n nodeB -n nodeB -n nodeC -n nodeD
```

このコマンドにより、次のような情報が出力されます。

```
Node Names:      nodeA
                  nodeB
                  nodeC
                  nodeD
```

Bridged networks (full probing performed):

```
1      lan3      (nodeA)
        lan4      (nodeA)
        lan3      (nodeB)
        lan4      (nodeB)

2      lan1      (nodeA)
        lan1      (nodeB)

3      lan2      (nodeA)
        lan2      (nodeB)

4      lan3      (nodeC)
        lan4      (nodeC)
        lan3      (nodeD)
        lan4      (nodeD)

5      lan1      (nodeC)
        lan1      (nodeD)

6      lan2      (nodeC)
        lan2      (nodeD)
```

IP subnets:

IPv4:

```
15.13.164.0      lan1      (nodeA)
                  lan1      (nodeB)

15.13.172.0      lan1      (nodeC)
                  lan1      (nodeD)

15.13.165.0      lan2      (nodeA)
                  lan2      (nodeB)

15.13.182.0      lan2      (nodeC)
                  lan2      (nodeD)

15.244.65.0      lan3      (nodeA)
                  lan3      (nodeB)

15.244.56.0      lan4      (nodeC)
                  lan4      (nodeD)
```

IPv6:

3ffe:1111::/64	lan3	(nodeA)
	lan3	(nodeB)
3ffe:2222::/64	lan3	(nodeC)
	lan3	(nodeD)

Possible Heartbeat IPs:

	15.13.164.0		
		15.13.164.1	(nodeA)
		15.13.164.2	(nodeB)
15.13.172.0		15.13.172.158	(nodeC)
		15.13.172.159	(nodeD)
15.13.165.0		15.13.165.1	(nodeA)
		15.13.165.2	(nodeB)
15.13.182.0		15.13.182.158	(nodeC)
		15.13.182.159	(nodeD)

Route connectivity(full probing performed):

1	15.13.164.0
	15.13.172.0
2	15.13.165.0
	15.13.182.0
3	15.244.65.0
4	15.244.56.0

[Route connectivity] セクションの左端の番号 (1~4) は、どのサブネットが相互に経路指定されているかを示しています (たとえば、15.13.164.0 と 15.13.172.0)。

- ❶ **重要:** この例では、NodeA と NodeB が使っているサブネット 15.244.65.0 は、NodeC と NodeD が使っているサブネット 15.244.56.0 へは経路指定されていません。

しかし、NodeA と NodeB が使っているサブネット 15.13.164.0 と 15.13.165.0 は、それぞれ、**NodeC** と NodeD が使っているサブネット 15.13.172.0 と 15.13.182.0 へ経路指定されています。cmquerycl が成功するためには、全ノード間に少なくとも 1 つこのような経路指定が存在しなければなりません。

クロスサブネット構成でのハートビートの構成については、「[クラスター構成のパラメーター](#)」(82 ページ)にある `HEARTBEAT_IP` パラメーターの説明を参照してください。

ハートビートサブネットの識別

クラスター構成ファイルには、ハートビートサブネット上の IP アドレスのエントリーが含まれています。専用のハートビートサブネットを使用するとともに、データサブネットを含む他のサブネットにハートビートを構成することをお勧めします。

ハートビートは、IPv4 サブネットまたは IPv6 サブネットに構成できます。

ハートビートはルーターで結合された複数の IPv4 サブネットで構成できます。この場合、各クラスターノードに対して少なくとも 2 つハートビートパスを構成しなければなりません。[HEARTBEAT_IP](#) (85 ページ) の説明と「[クロスサブネット構成](#)」(25 ページ) も参照してください。

構成できるパッケージの最大数を指定する

この値は、現在クラスターに構成されているパッケージ数以上の値でなければなりません。この数には、すべての種類のパッケージ (フェイルオーバーパッケージ、マルチノードパッケージ、システムマルチノードパッケージ) が含まれます。1 クラスターあたりのパッケージの最大数は 300 です。デフォルトはこの最大値です。

注記: カーネルパラメーターは、そのノードで同時に稼働するパッケージの最大数を十分に上回るように、各ノードで調整してください。

MEMBER_TIMEOUT パラメーターの変更

cmquerycl コマンドでは、`MEMBER_TIMEOUT` パラメーターのデフォルト値として 14 秒が指定されます。この値を変更すると、クラスターの再編成とフェイルオーバー回数が直接影響を受けます。システムの高負荷やネットワークトラフィックの過剰状態が原因となってクラスターノード障害が発生している場合は、必要に応じてこの値を増やしてください。また、クラスターの再編成に長い時間がかかる場合は、必要に応じてこの値を減らしてください。

`MEMBER_TIMEOUT` の変更は、クラスターの稼働中にも行えます。

ノードタイムアウトについての詳細は、「[ノードがタイムアウトしたときの動作](#)」(68 ページ)と、「[クラスター構成のパラメーター](#)」(82 ページ)にある `MEMBER_TIMEOUT` パラメーターの説明、および「[MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスターの再編成](#)」(244 ページ)を参照してください。

クラスターへのアクセスの制御

Serviceguard のアクセス制御ポリシーは、クラスターユーザーの管理権限や監視権限を定義します。

用語についての注意事項

Serviceguard コマンドの出力で Role-Based Access(RBA) という用語が表示されることもありますが、本書で優先的に使う用語は、次のとおりです。

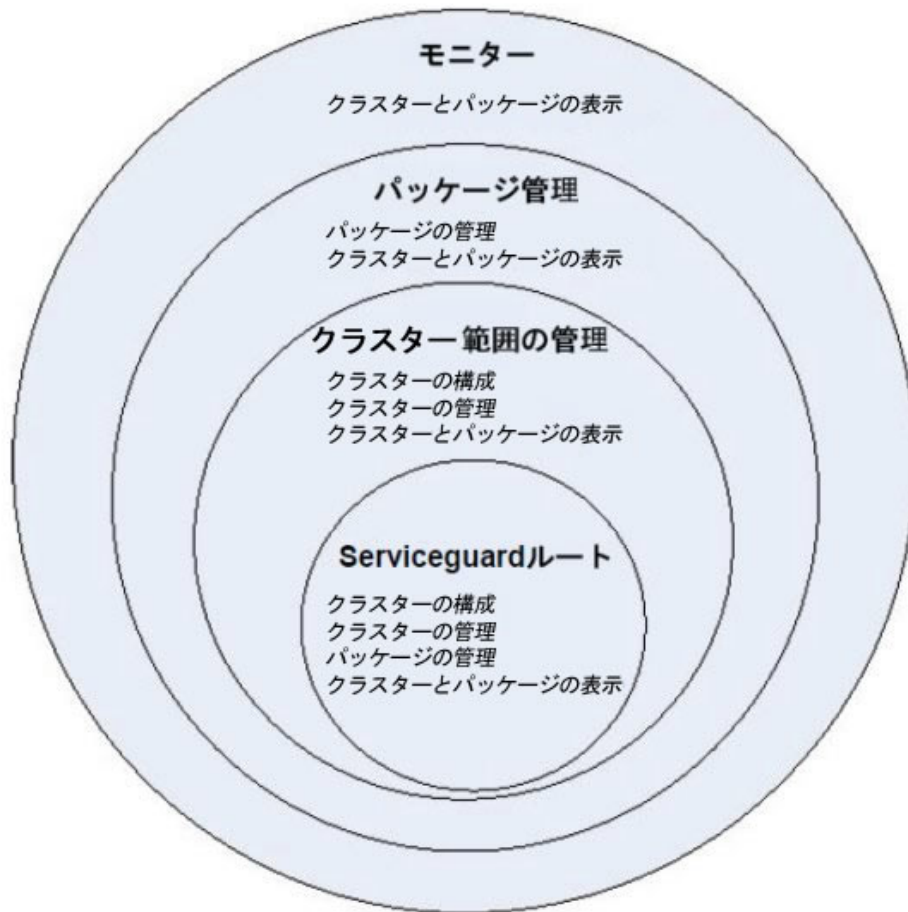
- アクセス制御ポリシー - クラスターへのユーザーアクセスを定義する規則のセット。
 - アクセス制御ポリシー - 3 つのパラメーター `USER_NAME`、`USER_HOST`、`USER_ROLE` からなる規則のいずれか。「[アクセス制御ポリシーの設定](#)」(148 ページ)を参照してください。
- アクセスロール - クラスターのユーザー用に定義可能なロールのセット (モニター、パッケージ管理、クラスター範囲の管理)。
 - アクセスロール - これらのロールのいずれか (例: モニター)。

アクセスロールの動作

Serviceguard デーモンは、コマンドユーザーのホスト名とユーザー名を、定義されているアクセス制御ポリシーと照合することで、Serviceguard コマンドへのアクセスを許可します。各ユーザーは、自分のロールで許可されているコマンドだけを実行できます。

次の図は、アクセスロールと、その権限を示しています。最も内側の円が、信頼度が最も高いロールです。最も外側が、信頼度が最も低いロールです。各ロールでは自分自身の機能と、自分より外側の機能をすべて実行できます。たとえば、「Serviceguard ルート」では、自分自身の機能と、「クラスター範囲の管理」、「パッケージ管理」、および「モニター」の機能をすべて実行できます。「クラスター範囲の管理」では、自分自身の機能と、「パッケージ管理」および「モニター」の機能をすべて実行できます。その他のロールも同様です。

図 27 アクセスロール



アクセスレベル

Serviceguard は、ルートアクセスと非ルートアクセスの 2 種類のアクセスレベルを認識します。

- ルートアクセス: **全機能。クラスターを構成できる唯一のロールです。**

図 27 に示すように、ルートアクセスのユーザーは、クラスターおよびパッケージの構成を全面的に制御できます。ルートアクセスは、コマンド `cmcheckconf`、`cmapplyconf`、`cmdeleteconf`、および `cmmodnet -a` の使用が許されている唯一のロールです。

この Serviceguard ロールを実行するためには、管理するクラスターのノードでルートユーザー (スーパーユーザー) としてログインしなければなりません。反対に、クラスター内のノードのルートユーザーは、そのクラスターの完全な Serviceguard ルートアクセス権限を常に保有しています。この権限を付与するために Serviceguard の構成を追加する必要はありません。

- ① **重要:** クラスター外部のシステムのユーザーは、セキュアな接続 (`rsh` または `ssh`) を使用する場合にのみ、クラスターを構成する Serviceguard ルートアクセス権限を取得できます。

- 非ルートアクセス: その他のユーザーには、以下の 4 つのロールのいずれかを割り当てることができます。
 - クラスター範囲の管理: **クラスターの管理、パッケージの管理、およびクラスターとパッケージの表示操作を実行することができます。**
このユーザーは、クラスターを管理することはできますが、クラスターの構成や作成はできません。「クラスター範囲の管理」には、パッケージ管理ロールの権限が含まれています。
 - (すべてのパッケージの)パッケージ管理: **パッケージの管理、およびクラスターとパッケージの表示コマンドを実行できます。**
このユーザーは、クラスター内の任意のパッケージの実行と停止、パッケージの切り替え動作の変更を行うことはできますが、パッケージの構成や作成はできません。単一パッケージの「パッケージ管理」とは異なり、このロールはクラスター構成ファイル内で定義されます。「パッケージ管理」には、クラスター範囲のモニターロールの権限が含まれています。
 - (単一パッケージの)パッケージ管理: **特定のパッケージのパッケージ管理を実行でき、クラスターとパッケージの表示コマンドを使うことができます。**
このユーザーは、指定されたパッケージの実行と停止、パッケージの切り替え動作の変更を行うことはできますが、パッケージの構成や作成はできません。これはパッケージ構成ファイル内で定義される唯一のアクセスロールです。他のロールは、クラスター構成ファイル内で定義されます。単一パッケージの「パッケージ管理」には、クラスター範囲のモニターロールの権限も含まれています。
 - モニター: **クラスターとパッケージの表示操作を実行できます。**
このユーザーは、クラスターとそのパッケージに対して読み取り専用のアクセスを行うことができます。
- ① **重要:** リモートユーザー (クラスター内のノードにログインしておらず、rsh または ssh による接続を行っていないユーザー) に許されるのは、クラスターに対するモニターアクセスだけです。
(クラスター範囲の管理とパッケージ管理をリモートユーザーに対して構成できますが、そのような使用法は推奨されません。Serviceguard A.11.18 から、クラスター範囲の管理またはパッケージ管理をリモートユーザー用に構成すると、モニターロールが付与されるようになりました。詳細は、「[アクセス制御ポリシーの設定](#)」(148 ページ)を参照してください。)

アクセス制御ポリシーの設定

各クラスターノードのルートユーザーは、すべてのノード上で自動的に Serviceguard ルートアクセスが許可されます。(詳細は、「[ルートレベルアクセスの構成](#)」(123 ページ)を参照してください。) アクセス制御ポリシーは、その他のクラスターユーザーの非ルートロールを定義します。

注記: 詳細情報とアドバイスについては、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] -> [White Papers] にあるホワイトペーパー『Securing Serviceguard』を参照してください。

クラスター構成ファイルで、クラスターのアクセス制御ポリシーを定義します。「[クラスター構成のパラメーター](#)」(82 ページ)を参照してください。特定のパッケージについてアクセス制御を定義するには、構成ファイル内の `user_host` (178 ページ) と関連パラメーターを使います。クラスターあたり最大 200 のアクセス制御ポリシーが定義できます。ルートユーザーは、クラスターの稼働中にアクセス制御ポリシーの作成や変更を行うことができます。

注記: ノードをクラスターに構成すると、クラスター構成ファイルおよびパッケージ構成ファイルに設定したアクセス制御ポリシーによってクラスター全体のセキュリティが制御されます。「ブートストラップ」ファイル `cmclnodelist` への変更は無視されます (「[未構成ノードへのルートアクセスの許可](#)」(123 ページ) を参照)。

アクセス制御ポリシーは、構成ファイル内で以下の 3 つのパラメーターを使って定義します。

- `USER_NAME` は、`ANY_USER`、またはホスト `USER_HOST` にある `/etc/passwd` ファイル内の最大 8 つのログイン名のいずれかです。名前は、次の例のように、スペースまたはタブで区切る必要があります。

```
# Policy 1:
USER_NAME john fred patrick
USER_HOST bit
USER_ROLE PACKAGE_ADMIN
```

- `USER_HOST` は、`USER_NAME` が Serviceguard コマンドを実行するノードです。

注記: コマンドは `USER_HOST` で実行する必要がありますが、他のノードに対する操作を実行できます。たとえば、patrick は bit のコマンド行で gryf 上のパッケージを起動できます (bit と gryf が同じクラスターの場合)。

`USER_HOST` には、以下の 3 つの値から 1 つを選びます。

- `ANY_SERVICEGUARD_NODE` - Serviceguard が構成され、このクラスター内のノードが通信可能な (`cmquerycl -w full` で出力される) サブネット上にあるノード。

注記: `USER_HOST` に `ANY_SERVICEGUARD_NODE` を設定した場合、`USER_ROLE` には `MONITOR` を設定してください。クラスター外から接続するユーザーに、これより高い権限を与えることはできません (ただし、`rsh` や `ssh` を介して接続する場合は除きます。この接続は、ローカル接続として扱われます)。

ネットワーク構成によっては、`ANY_SERVICEGUARD_NODE` により、クラスターに対する広範囲の読み取り専用アクセスが提供される可能性があります。

- `CLUSTER_MEMBER_NODE` - クラスター内の任意のノード。
- 特定のノード名 - 使用中の名前解決サービスで解決可能な完全修飾ドメイン名のホスト名要素 (最初の要素) を使用します。各ノードの `/etc/hosts` にも一致します。IP アドレスや完全修飾ドメイン名を使わないでください。1 つの IP アドレスに複数のホスト名が存在する (エイリアス) 場合は、いずれかのホスト名が `USER_HOST` に一致しなければなりません。詳細は、「[名前解決の構成](#)」(124 ページ) を参照してください。

- `USER_ROLE` は、以下の 3 つの値のいずれかでなければなりません。

- `MONITOR`
- `FULL_ADMIN`
- `PACKAGE_ADMIN`

`MONITOR` と `FULL_ADMIN` は、クラスター構成ファイルの中だけで設定することができ、クラスター全体に適用されます。`PACKAGE_ADMIN` は、クラスター構成ファイルまたはパッケージ構成ファイルで設定することができます。クラスター構成ファイルで設定した場合には、`PACKAGE_ADMIN` はすべての構成済みのパッケージに適用されます。パッケージ構成ファイルで設定した場合には、そのパッケージにのみ適用されます。これらのロールは排他的ではありません。たとえば、同じパッケージに対して、複数のユーザーに `PACKAGE_ADMIN` ロールを付与することができます。

注記: アクセス制御ポリシーを構成したり変更するために、クラスターやパッケージを停止する必要はありません。

アクセス制御ポリシーの例を示します。

```
USER_NAME john
USER_HOST bit
USER_ROLE PACKAGE_ADMIN
```

このポリシーがクラスター構成ファイルで定義されている場合には、ユーザーjohnに対して、ノードbitのすべてのパッケージのPACKAGE_ADMINロールが与えられます。また、PACKAGE_ADMINにはMONITORが含まれているため、ユーザーjohnにはクラスター全体のMONITORロールも与えられます。このポリシーがPackageAのパッケージ構成ファイルで定義されている場合、ノードbitのユーザーjohnには、PackageAに対してのみPACKAGE_ADMINロールが付与されます。

クラスターのロールを計画したら、できるだけ早めに検証してください。組織のセキュリティポリシーで許可されている場合は、グループログインを作成する方が簡単な場合があります。たとえば、CLUSTER_MEMBER_NODE(つまり、クラスター内の任意のノード)のユーザーoperator1にMONITORロールを割り当てます。そして、このログイン名とパスワードを、クラスターを監視するすべての人に知らせることができます。

ロールの競合

同じユーザーとホストに対して、異なるロールを構成しないでください。Serviceguardはこのような構成を競合として扱い、構成を適用する際にエラーで失敗します。ANY_USERやANY_SERVICEGUARD_NODEなどの「ワイルドカード」は例外です。ANY_USERとjohnに異なるロールを割り当てることはできます。

-
- ① **重要:** ワイルドカードで指定されているクラスの個々のメンバーに許可されている高いレベルのロールが、ワイルドカードによって下げられることはありません。たとえば、リモートシステムのルートユーザーがクラスターにアクセスできるように、次のポリシーを設定したとします。

```
USER_NAME root
USER_HOST ANY_SERVICEGUARD_NODE
USER_ROLE MONITOR
```

この設定により、このクラスターのノードにルートとしてログインしているユーザーのアクセスレベルが低くなることは**ありません**。このようなユーザーには、常に完全なServiceguardルートアクセス権限が与えられます。

以下のエントリーをクラスター構成ファイルに指定したとします。

```
# Policy 1:
USER_NAME john
USER_HOST bit
USER_ROLE PACKAGE_ADMIN

# Policy 2:
USER_NAME john
USER_HOST bit
USER_ROLE MONITOR

# Policy 3:
USER_NAME ANY_USER
USER_HOST ANY_SERVICEGUARD_NODE
USER_ROLE MONITOR
```

この例では、ユーザー john に 2 つのロールを割り当てているため失敗します。(いずれにしても、PACKAGE_ADMIN には MONITOR ロールがすでに含まれているため、Policy 2 は不要です。)

ワイルドカード ANY_USER には個別ユーザーの john が含まれていますが、Policy 3 は他のポリシーと競合しません。

注記: ANY_USER や ANY_SERVICEGUARD_NODE などのワイルドカードを入力する場合には、特に慎重にスペルを確認してください。スペルに誤りがあると、Serviceguard は個別のユーザーやノードであるとみなします。

パッケージとクラスターのロール

パッケージ構成とクラスター構成の間でロールに矛盾があると、パッケージの構成が失敗します。そのため、パッケージのロールを作成するときには、クラスター構成ファイルを参照することをお勧めします。クラスター構成ファイルのリストを出力するには、cmgetconf を使います。

ユーザー名またはホスト名に対してクラスター構成ファイルでロールが構成されている場合、パッケージ構成ファイルでは同じユーザー名またはホスト名に対してロールを指定しないでください。また、クラスターノードのルートユーザークラスターとそのパッケージの管理に対する完全な制御権をすでに持っているため、このユーザーにパッケージ管理ロールを割り当てても意味はありません。

クラスター構成の確認

クラスター構成テンプレートファイルを編集した場合は、ファイルの内容を確認するために次のコマンドを使用します。

```
cmcheckconf -v -C $SGCONF/clust1.conf
```

このコマンドは以下をチェックします。

- ネットワークのアドレスと接続。
- クォーラムサーバーの接続。
- すべてのノード上のすべてのロック LUN のデバイス名が同じ物理ディスク領域を参照していること。
- 各ノードでロックデバイスが指定されており、単一であること。
- クォーラムサーバーまたはロック LUN のいずれかが構成されていること (また、両方が構成されていないこと)。
- 名前が重複していないこと。
- コマンド行で指定したスクリプトの存在とそのパーミッション。
- 指定されたすべてのノードが同じハートビートサブネットにあること。
- 構成ファイル名が正しいこと。
- すべてのノードにアクセスできること。
- CLUSTER_NAME、MEMBER_TIMEOUT、AUTO_START_TIMEOUT が、複数指定されていないこと。
- パッケージの起動および停止スクリプトのタイムアウト値が最大値を超えないこと。
- AUTO_START_TIMEOUT 変数の値が、0 以上であること。
- ハートビートネットワークの必要最小限の要件「[クラスター構成のパラメーター](#)」(82 ページ)の HEARTBEAT_IP を参照してください。
- NODE_NAME が 1 つ以上指定されていること。
- 各ノードが、それぞれのハートビートネットワークに接続されていること。

- すべてのハートビートネットワークが、同じ種類の LAN であること。
- 指定されたネットワークインターフェイスのデバイスファイルが、有効な LAN デバイスファイルであること。
- クラスターのその他の構成パラメーターおよびパッケージが有効であること。

クラスターがオンラインの場合は、`cmcheckconf` コマンドで構成上の個々の変更が条件を満足しているのかも確認されます。

クラスターロック構成メッセージ

`cmquerycl`、`cmcheckconf`、`cmapplyconf` の各コマンドは、クラスターロックが正しく構成されていない場合にエラーを返します。2 ノードのクラスター構成でクラスターロックがない場合には、クラスター構成ファイルに以下のメッセージが出力されます。

```
# Warning: Neither a quorum server nor a lock lun was specified.
# A Quorum Server or a lock lun is required for clusters of only two nodes.
```

クォーラムサーバーとロック LUN の両方を構成しようとする、`cmcheckconf` コマンドまたは `cmapplyconf` コマンドの発行時に、標準出力に以下のメッセージが表示されます。

```
Duplicate cluster lock, line 55. Quorum Server already specified.
```

バイナリ構成ファイルの配布

クラスターのパラメーターをすべて指定した後、`cmapplyconf` コマンドを使用して、構成を適用します。この作業により、クラスター内のすべてのノードに対してバイナリ構成ファイルが配布されます。この作業は、パッケージを構成する**以前に**、あらかじめ済ませておくといでしょう (次章を参照)。このようにすると、稼働中のクラスターに対して `cmviewcl` コマンドを実行して、クォーラムサーバー、ハートビートネットワーク、その他のクラスターレベルの動作を確認できます。構成を配布する前に、クラスターノード間でセキュリティファイルのコピーが許可されていることを確かめておきます。[「ルートレベルアクセスの構成」\(123 ページ\)](#) を参照してください。

次のコマンドで、バイナリ形式構成ファイルを配布します。

```
cmapplyconf -v -C $SGCONF/clust1.conf
```

稼働中のクラスターの管理

この項では、クラスターの管理を定型化する方法について説明します。詳細は、[第7章「クラスターとパッケージの管理」\(186 ページ\)](#) を参照してください。クラスターの管理は、Serviceguard Manager から行うことも、下記のように Serviceguard コマンドで行うこともできます。

Serviceguard コマンドでのクラスターの動作チェック

- `cmviewcl` は、クラスターとその多数の構成要素のステータスを確認するのに使います。モニターロールが割り当てられた非ルートユーザーは、クラスターノードでこのコマンドを実行することや、Serviceguard Manager でステータス情報を参照することができます。
- `cmrunnode` は、ノードを起動するのに使用します。クラスター範囲の管理ロールが割り当てられた非ルートユーザーは、クラスターノードまたは Serviceguard Manager からこのコマンドを実行できます。
- `cmhaltnode` は、稼働中のノードを手動で停止するのに使います (このコマンドは、`shutdown(1m)` でも使用されます)。クラスター範囲の管理ロールが割り当てられた非ルートユーザーは、クラスターノードまたは Serviceguard Manager からこのコマンドを実行できます。
- `cmruncl` は、停止しているクラスターを手動で起動するのに使います。クラスター範囲の管理ロールが割り当てられた非ルートユーザーは、クラスターノードまたは Serviceguard Manager からこのコマンドを実行できます。

- `cmhaltcl` は、クラスターを手動で停止するのに使います。クラスター範囲の管理ロールが割り当てられた非ルートユーザーは、クラスターノードまたは Serviceguard Manager からこのコマンドを実行できます。

これらのコマンドを使用して、以下のようにクラスターの動作をテストできます。

1. クラスターが実行されていない場合は開始します。

`cmruncl -v`

`cmruncl` は、デフォルトでは、ネットワークのチェックを行います。Serviceguard は、クラスター構成のネットワーク情報をもとにして実際のネットワーク構成をチェックします。この検証が不要な場合には、代わりに `cmruncl -v -w none` を使って検証を行わないようにし、時間を節約することができます。

2. クラスターが起動したら、クラスターの構成要素が正常に動作していることを確認します。

`cmviewcl -v`

すべてのノードとネットワークが期待どおりに動作していることを確認します。詳細は、「クラスターとパッケージの管理」についての章を参照してください。

3. 以下の手順を実行して、クラスターでのノードの切り離しや組み込みが正しく実行されることを確かめます。
 - クラスターを停止します。Serviceguard Manager を使うことも、`cmhaltnode` コマンドを使うこともできます。
 - クラスターのメンバー構成を確認し、ノードがクラスターから切り離されたことを確かめます。Serviceguard Manager のメインページまたは `cmviewcl` コマンドを使うこともできます。
 - ノードを起動します。Serviceguard Manager を使うことも、`cmrunnode` コマンドを使うこともできます。
 - ノードが再び稼働していることを確認します。Serviceguard Manager を使うことも、`cmviewcl` コマンドを使うこともできます。
4. クラスターを停止させます。Serviceguard Manager を使うことも、`cmhaltcl -v -f` コマンドを使うこともできます。

これらのコマンドの詳細はマンページを参照してください。クラスターテストについての詳細は、[第8章「クラスターのトラブルシューティング」\(234 ページ\)](#)を参照してください。

自動起動機能のセットアップ

自動起動とは、各ノードが個別にクラスターに結合するプロセスです。Serviceguard には、起動プロセスを制御する起動スクリプトがあります。各ノードは、クラスターがすでに存在する場合はそのクラスターへの結合を試み、クラスターが稼働していなければ、すでに構成されている全ノードを含むクラスターの形成を試みます。クラスターの起動では、クラスターの自動起動を使うことをお勧めします。システム管理者が操作する必要はありません。

以下の 3 つの場合があります。

- クラスターはどのノードでも稼働しておらず、すべてのクラスターノードが到達可能で、すべてのノードが起動しようとしている場合。この場合、ノードは構成済みのすべてのノードから成るクラスターの形成を試みます。
- クラスターが少なくとも 1 台のノードですでに稼働している場合。この場合、ノードはそのクラスターへの結合を試みます。
- どちらでもない場合。すなわち、クラスターがどのノードでも動作しておらず、到達可能なノードや起動しようとしていないノードがある場合。この場合は、ノードは `AUTO_START_TIMEOUT` の間起動を試みます。その間に上記の条件のいずれも満たされないときには、起動に失敗します。

クラスターを自動起動する場合は、クラスター内の各ノード上で \$SGAUTOSTART ファイル (\$SGCONF/cmcluster.rc) の AUTOSTART_CMCLD フラグを 1 に設定します。設定されたノードは、ブート時にクラスターに自動的に結合されます。

\$SGAUTOSTART ファイルの例を示します。

```
SGAUTOSTART=/usr/local/cmcluster/conf/cmcluster.rc
#***** CMCLUSTER *****

# Highly Available Cluster configuration
#
# @(#) $Revision: 82.2 $
#
#
# AUTOSTART_CMCLD
#
# Automatic startup is the process in which each node individually
# joins a cluster. If a cluster already exists, the node attempts
# to join it; if no cluster is running, the node attempts to form
# a cluster consisting of all configured nodes. Automatic cluster
# start is the preferred way to start a cluster. No action is
# required by the system administrator. If set to 1, the node will
# attempt to join/form its CM cluster automatically as described
# above. If set to 0, the node will not attempt to join its CM
# cluster.

AUTOSTART_CMCLD=1
```

注記: /sbin/init.d/cmcluster ファイルは、Serviceguard が \$SGCONF/rc に格納しているファイル呼び出します。(Linux の各ディストリビューションにおける Serviceguard ディレクトリの位置については、「[Serviceguard のファイルの位置](#)」(122 ページ)を参照してください。) このディレクトリは Serviceguard 専用です。このディレクトリでファイルの移動、削除、変更、追加は行わないでください。

システムメッセージの変更

システムのログインメッセージに以下のような内容を追加すると便利です。

```
This system is a node in a high availability cluster.
Halting this system may cause applications and services to
start up on another node in the cluster.
```

またログインメッセージには、全クラスターノードの一覧やクラスターに固有の情報を追加するのもよいでしょう。

/etc/motd ファイルをカスタマイズすると、このようなクラスター情報をメッセージに追加できます。

単一ノードクラスターの管理

クラスターに必要なノード数は、保護の対象となるアプリケーションの処理能力がどの程度求められるかにより異なります。

単一ノードクラスターでは、クラスター内に他のノードがないため、クォーラムサーバーは必要ありません。cmquerycl コマンドからの出力では、ノードが 1 つしかない場合には、クォーラムサーバー情報のエリアが省略されます。

ネットワークには冗長性を持たせる必要がありますが、ハートビートを送信する相手のノードがないため、ハートビート LAN を指定する必要はありません。クラスター構成ファイルでは、Serviceguard でモニターする LAN をすべて指定します。すでに IP アドレスのある LAN については、HEARTBEAT_IP パラメーターではなく、STATIONARY_IP パラメーターを付けて IP アドレスを指定します。

単一ノードによる運用

単一ノードによる運用は、単一ノードクラスターや、1つを除いてすべてのノードがダウンしたりシャットダウンされたマルチノードクラスターで行われます。このとき、稼働しているノード上ではアプリケーションが実行されている可能性があります。Serviceguard のデーモン cmcld がアクティブである限り、後で他のノードをクラスターに再結合させることは可能です。

単一ノードでの運用中に、cmcld デーモンが異常終了した場合は、ノードは稼働したまま、アプリケーションも実行中のままとなります (これは、マルチノードクラスターで cmcld が異常終了した場合とは異なります。マルチノードクラスターでは、ノードはリブートにより停止し、パッケージは引き継ぎノードに切り替えられます)。

このケースでは、アプリケーションはまだ動作しており、パッケージ切り替えに対応できるノードもないため、単一ノードを停止する必要はありません。

- △ **注意:** ただし、Serviceguard の再起動は行わないでください。単一ノードでまだ動作しているアプリケーションの新しいインスタンスが別のノードで起動されると、データが破損するおそれがあるためです。代わりに、適当なタイミングを見計らってノードをシャットダウンした後、リブートを行ってください。これによりアプリケーションがシャットダウンされるため、Serviceguard はリブート後にクラスターを再起動できるようになります。

identd の無効化

identd を無効にする必要がある場合以外は、この項を無視してください。

Serviceguard を、identd を使わないように構成することができます。

- △ **注意:** この設定はお勧めしません。詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] -> [White Papers] にあるホワイトペーパー『Securing Serviceguard』を参照してください。

identd を無効にする必要がある場合は、Serviceguard のインストール後、各ノードがクラスターに再結合する前 (たとえば、cmrunnode または cmruncl を実行する前) に、各ノード上で次の手順を実行します。

Red Hat および SUSE の場合:

1. /etc/xinetd.d/haci-cfg ファイルの `server_args` パラメーターの値を、`-c` から `-c -i` に変更します。
2. xinetd を再起動します。

```
/etc/init.d/xinetd restart
```

クラスター構成の削除

cmdeleteconf コマンドを実行すると、クラスター構成を削除できます。コマンドを入力すると、ファイルの削除を確認するプロンプトが表示されます。ただし `-f` オプションを指定した場合は、プロンプトは表示されません。構成を削除できるのは、クラスターが停止している間だけです。構成を削除すると、クラスター内の全ノードからバイナリ構成ファイルが削除され、削除以前にクラスターに認識されていたボリュームグループはすべて認識されなくなります。

注記: cmdeleteconf コマンドは、クラスターバイナリファイル `$SGCONF/cmc1config` だけを削除します。`$SGCONF` ディレクトリ内のその他のファイルは一切削除しません。

クラスター構成はクラスターが停止中でなければ削除できませんが、cmdeleteconf コマンドは、クラスター内の全ノードの電源が入っていてノードにアクセスできる状態で実行します。ノードの電源が入っていない場合は、電源を入れ、ブートできるようにください。アクセスできないノードがあれば、その一覧と以下のメッセージが表示されます。

```
Checking current status
cmdeleteconf: Unable to reach node lpctest1.
WARNING: Once the unreachable node is up, cmdeleteconf
should be executed on the node to remove the configuration.
```

```
Delete cluster lpcluster anyway (y/[n])?
```

メッセージに対して **yes** を選択すると、構成が削除されます。その後、アクセスできなかったノードにアクセスできるようになったら、そのノード上で `cmdeleteconf` コマンドを実行して構成ファイルを削除してください。

6 パッケージとサービスの構成

Serviceguard パッケージは、アプリケーションおよびサービスと、それが依存するリソースをグループ化したものです。

典型的な Serviceguard パッケージは、1 台のノードで起動し、必要に応じて別のノードに移動 (フェイルオーバー) 可能なフェイルオーバーパッケージです。詳細は、「[Serviceguard for Linux について](#)」(18 ページ)、「[パッケージマネージャーの動作](#)」(38 ページ)、および「[パッケージ構成のプランニング](#)」(92 ページ) を参照してください。

マルチノードパッケージも作成できます。これは複数のノードで同時に動作するパッケージです。

クラスター内の全ノードで動作するシステムマルチノードパッケージがサポートされるのは、当社提供のアプリケーションだけです。

パッケージの作成や変更では、以下の手順を実行する必要があります。これらの手順は、以降の項で説明されています。

1. パッケージの主要な特性を判断し、含める必要があるモジュールを選択します([157 ページ](#))。
2. パッケージ構成ファイルを作成します([179 ページ](#))。
3. 構成ファイルを編集します([181 ページ](#))。
4. パッケージ構成を検証し、適用します([184 ページ](#))。
5. パッケージをクラスターに追加します([184 ページ](#))。

注記: これは、Serviceguard A.11.18 からの新しいパッケージ構成プロセスです。本書では、この方法で作成されたパッケージをモジュラーパッケージと呼び、新しいパッケージはこれを使って作成されると想定しています。この方法は、以前の方法より簡単で効率的です。パッケージは小さなモジュールから構築でき、独立したパッケージ制御スクリプトが不要で、手作業で配布する必要もありません。

Serviceguard A.11.16 以前を使って作成されたパッケージを従来のパッケージと呼びます。新しいパッケージを作成するのではなく、従来のパッケージの再構成が必要な場合には、「[従来のパッケージの構成](#)」([218 ページ](#)) を参照してください。

「[従来のパッケージの構成](#)」で説明されている方法で、従来のパッケージを新たに作成することもできます。Serviceguard Toolkit を使用する場合は、それぞれの製品のドキュメントを参照してください。

従来のパッケージをモジュラーパッケージに変換する場合は、「[従来のパッケージからモジュラーパッケージへの移行](#)」([226 ページ](#)) を参照してください。Serviceguard Toolkit パッケージは変換しないでください。

(従来のパッケージではパッケージ制御スクリプトに含まれるが、モジュラーパッケージではパッケージ構成ファイルに含まれるようになったパラメーターは、このセクションの後半に含まれる表「[オプションのパッケージモジュール](#)」([160 ページ](#)) ではマーク (S) を付けて示しています。)

パッケージモジュールの選択

- ❶ **重要:** 作業を開始する前に、「[パッケージ構成のプランニング](#)」([92 ページ](#)) で説明しているパッケージプランニング作業を行う必要があります。

適切なパッケージモジュールを選択するためには、作成するパッケージに関して以下の項目を決定する必要があります。

- パッケージのタイプ。「[パッケージのタイプ: フェイルオーバー、マルチノード、システムマルチノード](#)」([158 ページ](#)) を参照してください。

- パッケージに指定する必要があるパラメーター (フェイルオーバー、マルチノード、システムマルチノードなどのベースタイプに含まれているパラメーター以外のパラメーター)。「[パッケージモジュールとパラメーター](#)」(159 ページ)を参照してください。

以上の項目が決まれば、パッケージ構成ファイルの生成の準備が整ったことになります。「[パッケージ構成ファイルの生成](#)」(179 ページ)を参照してください。

パッケージのタイプ: フェイルオーバー、マルチノード、システムマルチノード

パッケージには 3 つの種類があります。

- フェイルオーバーパッケージ。これは最も一般的なパッケージタイプです。フェイルオーバーパッケージは、一度に 1 つのノードで稼働します。障害が発生すると、Serviceguard (またはユーザー) はパッケージを停止し、パッケージの構成リスト (`node_name` (164 ページ)を参照) から選択した別のノードでパッケージを再起動します。

フェイルオーバーパッケージを作成するためのパッケージ構成ファイルを生成するには、`cmmakepkg` コマンドの行で `-m sg/failover` を指定します。「[パッケージ構成ファイルの生成](#)」(179 ページ)を参照してください。

- マルチノードパッケージ。このパッケージは、クラスター内の複数のノードで同時に動作します。アプリケーション、サービス、汎用リソースやサブネットなどのパッケージコンポーネントで障害が発生すると、障害が発生したノード上のパッケージが停止するだけです。

マルチノードパッケージに再配置可能 IP アドレスを割り当てることはできません。

- ① **重要:** マルチノードパッケージは、Red Hat GFS(Red Hat GFS は Serviceguard A.11.20.00 ではサポートされません) などのクラスターファイルシステムを使うか、共有ストレージを使わないかのどちらかである必要があります。

マルチノードパッケージを作成するためのパッケージ構成ファイルを生成するには、`cmmakepkg` コマンドの行で `-m sg/multi_node` を指定します。「[パッケージ構成ファイルの生成](#)」(179 ページ)を参照してください。

- システムマルチノードパッケージ。システムマルチノードパッケージは、当社が提供するアプリケーションでのみサポートされます。

注記: マルチノードパッケージでは、以下のパラメーターは構成できません。

- `failover_policy`
- `failback_policy`
- `ip_subnet`
- `ip_address`

このタイプのパッケージに構成するボリュームグループは、共有モードでアクティブ化する必要があります。

パッケージのタイプと動作についての詳細は、「[パッケージマネージャーの動作](#)」(38 ページ)を参照してください。パッケージのプランニングについての詳細は、「[パッケージ構成のプランニング](#)」(92 ページ)を参照してください。

作成するパッケージのタイプが決まったら、次の手順は、含めるパッケージ構成モジュールを決めることです。「[パッケージモジュールとパラメーター](#)」(159 ページ)を参照してください。

フェイルオーバーパッケージとマルチノードパッケージの違い

マルチノードパッケージとフェイルオーバーパッケージの動作の主な違いを以下に示します。

- マルチノードパッケージの `auto_run` を無効にする (パッケージ構成ファイルで `no` に設定する) と、クラスターの起動時にそのマルチノードパッケージは起動しません。最初は、`cmmmodpkg` を使ってパッケージ切り替えを有効にすることで、パッケージを起動できま

す。しかし、その後 `cmhaltpkg` を使ってマルチノードパッケージを停止した場合、そのマルチノードパッケージは (`cmmodpkg` ではなく) `cmrunpkg` でしか再起動できません。

- マルチノードパッケージを `cmhaltpkg` を使って停止した場合、パッケージ切り替えは無効になりません。つまり、再起動した特定のノードで実行されるようにパッケージを構成すると、停止したそのパッケージは、依存性が満たされる限り、再起動したそのノードで実行を開始します。
- マルチノードパッケージを初めて起動する (クラスターの起動時、または `auto_run` が「no」に設定されている場合は、クラスターの起動に続いてパッケージ切り替えを有効にする) と、依存しているパッケージがそのプライマリノードで起動します。しかし、マルチノードパッケージを依存しているパッケージとともに停止し、そのマルチノードパッケージを再起動した場合、パッケージ切り替えを再度有効に設定した依存パッケージはそのマルチノードパッケージのインスタンスを実行できる最初のノードで起動します。これは、依存しているパッケージのプライマリノードとは限りません。

フェイルオーバーパッケージと依存関係のあるマルチノードパッケージの再起動が必要になった場合、依存する側であるフェイルオーバーパッケージを必ずそのプライマリノード上で再起動させるためには、マルチノードパッケージが再起動するまで依存する側のフェイルオーバーパッケージのパッケージ切り替えが再度有効にならないようにします。つまり、マルチノードパッケージの起動が完了した後でフェイルオーバーパッケージのパッケージ切り替えを有効にします。あるいは、`cmrunpkg` を使って、起動させるノードを指定して依存する側のフェイルオーバーパッケージを再起動することもできます。

パッケージモジュールとパラメーター

以下の表は、パッケージモジュールと、各モジュールに含まれる構成パラメーターを示します。この項は、「[パッケージ構成のプランニング](#)」(92 ページ) の説明と併せてお読みください。

ここでの説明と以降のパラメーターの説明(162 ページ) を参考にして、パッケージを作成するためにフェイルオーバーモジュール、マルチノードモジュール、システムマルチノードモジュールに追加するモジュールを決定します。従来のパッケージの作成に慣れている場合には、パッケージ制御スクリプトで使っていたパラメーター (またはそれと同等のもの) がパッケージ構成ファイルに移動していることが分かります。これらのパラメーターには、表では (S) が付けられています。

`cmmakepkg -l` (英字の「l」) を使うと、Serviceguard 用でないモジュール (HP Toolkit で提供されるモジュールなど) も含め、利用可能なモジュールをすべて表示できます。

注記: 多くのモジュールを含む複雑なパッケージを作成する場合には、モジュールの選択を省略して、単純にすべてのモジュールを含む構成ファイルを作成することができます。

```
cmmakepkg -m sg/all $SGCONF/pkg_sg_complex
```

(出力は `$SGCONF/pkg_sg_complex` に書き込まれます。)

ベースパッケージモジュール

`cmmakepkg` コマンド行で、少なくとも 1 つのベースモジュール (またはベースモジュールを含む `default` または `all`) を指定する必要があります。アスタリスク (*) が付いているパラメーターは、Serviceguard A.11.18、A.11.19、A.11.20.00 で新たに導入されたか、変更されたパラメーターです。(S) は、パッケージ制御スクリプトからモジュラーパッケージ用のパッケージ構成ファイルに移されたパラメーター (またはそれと同等のもの) を示します。詳細は、「[パッケージのパラメーターの説明](#)」(162 ページ) を参照してください。

表 8 ベースモジュール

モジュール名	パラメーター (ページ)	コメント
failover	<i>package_name</i> (163 ページ) * <i>module_name</i> (163 ページ) * <i>module_version</i> (163 ページ) * <i>package_type</i> (163 ページ) <i>package_description</i> (164 ページ) * <i>node_name</i> (164 ページ) <i>auto_run</i> (164 ページ) <i>node_fail_fast_enabled</i> (164 ページ) <i>run_script_timeout</i> (165 ページ) <i>halt_script_timeout</i> (165 ページ) <i>successor_halt_script_timeout</i> (166 ページ) <i>script_log_file</i> (166 ページ) <i>operation_sequence</i> (166 ページ) * <i>log_level</i> (166 ページ) * <i>failover_policy</i> (166 ページ) <i>failback_policy</i> (167 ページ) <i>priority</i> (167 ページ) *	ベースモジュール。フェイルオーバーパッケージの主要ビルディングブロックとして使います。 <i>package_type</i> (163 ページ) が <i>multi_node</i> または <i>system_multi_node</i> の場合、使用できません。
multi_node	<i>package_name</i> (163 ページ) * <i>module_name</i> (163 ページ) * <i>module_version</i> (163 ページ) * <i>package_type</i> (163 ページ) <i>node_name</i> (164 ページ) <i>auto_run</i> (164 ページ) <i>node_fail_fast_enabled</i> (164 ページ) <i>run_script_timeout</i> (165 ページ) <i>halt_script_timeout</i> (165 ページ) <i>successor_halt_timeout</i> (166 ページ) * <i>script_log_file</i> (166 ページ) <i>operation_sequence</i> (166 ページ) * <i>log_level</i> (166 ページ) * <i>priority</i> (167 ページ) *	ベースモジュール。マルチノードパッケージの主要ビルディングブロックとして使います。 <i>package_type</i> (163 ページ) が <i>failover</i> または <i>system_multi_node</i> の場合、使用できません。
system_multi_node	<i>package_name</i> (163 ページ) * <i>module_name</i> (163 ページ) * <i>module_version</i> (163 ページ) * <i>package_type</i> (163 ページ) <i>node_name</i> (164 ページ) <i>auto_run</i> (164 ページ) <i>node_fail_fast_enabled</i> (164 ページ) <i>run_script_timeout</i> (165 ページ) <i>halt_script_timeout</i> (165 ページ) <i>successor_halt_timeout</i> (166 ページ) * <i>script_log_file</i> (166 ページ) * <i>operation_sequence</i> (166 ページ) * <i>log_level</i> (166 ページ) * <i>priority</i> (167 ページ) *	ベースモジュール。システムマルチノードパッケージの主要ビルディングブロックとして使います。システムマルチノードパッケージは、当社が提供するアプリケーションでのみサポートされます。

オプションのパッケージモジュール

追加的な機能を構成する必要がある場合には、オプションのモジュールをベースモジュールに追加します。アスタリスク (*) が付いているパラメーターは、Serviceguard A.11.18、A.11.19、A.11.20.00 で新たに導入されたか、変更されたパラメーターです。(S) は、パッケージ制御スクリプトからモジュラーパッケージ用のパッケージ構成ファイルに移されたパラメーター (またはそれと同等のもの) を示します。詳細は、「[パッケージのパラメーターの説明](#)」(162 ページ) を参照してください。

表 9 オプションモジュール

モジュール名	パラメーター (ページ)	コメント
dependency	<i>dependency_name</i> (167 ページ) * <i>dependency_condition</i> (168 ページ) <i>dependency_location</i> (168 ページ)	ベースモジュールに追加して、他の 1 つ以上のパッケージに依存するパッケージを作成します。
weight	<i>weight_name</i> (168 ページ) * <i>weight value</i> (168 ページ) *	ベースモジュールに追加して、ノードのキャパシティに対して考慮される重みが定義されたパッケージを作成します。
monitor_subnet	<i>monitored_subnet</i> (169 ページ) * <i>monitored_subnet_access</i> (169 ページ) *	ベースモジュールに追加して、パッケージのサブネット監視を構成します。
package_ip	<i>ip_subnet</i> (170 ページ) * (S) <i>ip_subnet_node</i> (171 ページ) * <i>ip_address</i> (171 ページ) * (S)	failover モジュールに追加して、フェイルオーバーパッケージに再配置可能 IP アドレスを割り当てます。
service	<i>service_name</i> (171 ページ) * (S) <i>service_cmd</i> (171 ページ) (S) <i>service_restart</i> (172 ページ) * (S) <i>service_fail_fast_enabled</i> (172 ページ) <i>service_halt_timeout</i> (172 ページ)	ベースモジュールに追加して、アプリケーションまたはサービスを実行するパッケージを作成します。
generic_resource	<i>generic_resource_name</i> (172 ページ) <i>generic_resource_evaluation_type</i> (173 ページ) <i>generic_resource_up_criteria</i> (173 ページ)	ベースモジュールに追加して、汎用リソースを含むパッケージを作成します。汎用リソースは、カスタムモニターをユーザー定義のサービスとして構成することで、クリティカルリソースをこれらのモニターを介して監視するために使用できます。
volume_group	<i>vgchange_cmd</i> (174 ページ) * (S) <i>vg</i> (174 ページ) (S)	パッケージで LVM ボリュームにファイルシステム (Red Hat GFS 以外) をマウントする必要がある場合に、ベースモジュールに追加します。 Red Hat GFS は Serviceguard A.11.20.00 ではサポートされません。
filesystem	<i>concurrent_fsck_operations</i> (175 ページ) (S) <i>fs_mount_retry_count</i> (175 ページ) (S) <i>fs_umount_retry_count</i> (175 ページ) * (S) <i>fs_name</i> (176 ページ) * (S) <i>fs_directory</i> (176 ページ) * (S) <i>fs_type</i> (176 ページ) (S) <i>fs_mount_opt</i> (177 ページ) (S) <i>fs_umount_opt</i> (177 ページ) (S) <i>fs_fsck_opt</i> (177 ページ) (S)	ベースモジュールに追加して、パッケージのファイルシステムオプションを構成します。
pev	<i>pev_</i> (178 ページ) *	ベースモジュールに追加して、外部スクリプトに渡される環境変数を構成します。
external_pre	<i>external_pre_script</i> (178 ページ) *	ベースモジュールに追加して、パッケージ起動時のボリュームグループのアクティブ化の前と、パッケージ停止時のボリュームグループの非アクティブ化後に実行される追加プログラムを指定します。

表 9 オプションモジュール (続き)

モジュール名	パラメーター (ページ)	コメント
external	<code>external_script</code> (178 ページ) *	ベースモジュールに追加して、パッケージの起動時と停止時に実行される追加プログラムを指定します。
acp	<code>user_name</code> (178 ページ) <code>user_host</code> (178 ページ) <code>user_role</code> (179 ページ)	ベースモジュールに追加して、パッケージのアクセス制御ポリシーを構成します。
all	すべてのパラメーター	大部分のオプションパラメーターまたはすべてのオプションパラメーターを必要とする複雑なパッケージを作成する場合、またはすべての利用可能なパラメーターの仕様とコメントを表示する場合に使用します。
multi_node_all	マルチノードパッケージで利用できるすべてのパラメーター。これには、 <code>multi_node</code> 、 <code>dependency</code> 、 <code>monitor_subnet</code> 、 <code>service</code> 、 <code>volume_group</code> 、 <code>filesystem</code> 、 <code>pev</code> 、 <code>external_pre</code> 、 <code>external</code> 、および <code>acp</code> の各モジュールが含まれます。	このタイプのパッケージで使えるオプションパラメーターのほとんどまたはすべてを必要とするマルチノードパッケージを作成する場合に使用します。
default	(すべてのパラメーター)	<code>all</code> モジュールへのシンボリックリンク。 <code>cmmakepkg</code> コマンド行でベースモジュールが指定されなかった場合に使われます。「 cmmakepkg の例 」(180 ページ)を参照してください。
pr_cntl		パッケージで Persistent Reservation を有効にするベースモジュールを追加します。

注記: モジュラーパッケージ構成ファイルのパラメーター名のデフォルトの形式は小文字です。従来のパッケージではデフォルトは大文字です。互換性に関する問題はありません。Serviceguard は、少なくともパラメーター名については、大文字と小文字を区別しないためです。本書では、パラメーターが従来のパッケージでのみ使われる場合や、前後関係から従来のパッケージだけを対象にしていることがわかる場合を除き、パラメーター名には小文字を使用します。

パッケージのパラメーターの説明

ここでは、パッケージの構成パラメーターを簡単に説明します。

注記: 詳細は、`cmmakepkg` コマンドで出力される編集可能な構成ファイル内のコメント、および `cmmakepkg` (1m) のマンページを参照してください。

これらの説明を参照しながら必要なパラメーターを選択できるように、すべてのパラメーターのコメントを含む構成ファイルを生成して印刷することができます。このファイルを作成するには、次のコマンドを実行します。

```
cmmakepkg -m sg/all $SGCONF/sg-all
```

または、単純に次のコマンドを実行します。

```
cmmakepkg $SGCONF/sg-all
```

これにより、すべてのパラメーターとコメントを含む `$SGCONF/sg-all` ファイルが作成されます。(使用しているバージョンの Linux における `$SGCONF` の場所については、「[Serviceguard のファイルの位置](#)」(122 ページ)を参照してください。)

`cmmakepkg` の詳しい実行方法は、次項、「[パッケージ構成ファイルの生成](#)」(179 ページ)で説明します。

「[パッケージ構成のプランニング](#)」(92 ページ)も参照してください。

`package_name`

最大 39 文字の任意の名前ですが、以下の制限があります。

- 最初と最後の文字は英数字
- それ以外では、英数字とドット (.)、ダッシュ (-)、アンダースコア (_) のみを含む
- クラスタ内の他のパッケージ名と重複しない

- ① **重要:** Serviceguard の以前のリリースでのパッケージ名の制限は、これよりも緩いものでした。上記の規則に準拠していない名前を持つパッケージでも実行できますが、再構成時には名前を変更する必要があります。`cmcheckconf` と `cmapplyconf` では、新しい規則が強制されます。
-

`module_name`

モジュール名です。このパラメーターは変更しないでください。`cmmakepkg` のパラメーターとして相対パス (たとえば、`sg/failover`) の形式で使われ、パッケージの構成で使われるモジュールを指定します。(このファイルは、`$SGCONF/modules` ディレクトリにあります。使用しているバージョンの Linux における `$SGCONF` の場所については、「[Serviceguard のファイルの位置](#)」(122 ページ)を参照してください。)

モジュラーパッケージで新たに導入されました。

`module_version`

モジュールバージョンです。このパラメーターは変更しないでください。

モジュラーパッケージで新たに導入されました。

`package_type`

タイプは、`failover`、`multi_node`、`system multi_node` のいずれかです。ユーザーが構成できるのは、フェイルオーバーパッケージとマルチノードパッケージだけです。「[パッケージのタイプ: フェイルオーバー、マルチノード、システムマルチノード](#)」(158 ページ)を参照してください。

あるタイプのパッケージに別のタイプのベースモジュールを含めることはできません。たとえば、`package_type` が `failover` の場合、そのパッケージに `multi_node` または `system_multi_node` モジュールを含めることはできません。

package_description

パッケージが実行するアプリケーション。これは説明用のパラメーターであり、最大 80 文字を使って任意の値に設定できます。デフォルト値は、Serviceguard Package です。

node_name

このパッケージを実行できるノードです。または優先順位に従って並べられたノードのリストです。またはすべてのノードを示すアスタリスク (*) です。デフォルトは、* です。システムマルチノードパッケージの場合には、node_name * を指定する必要があります。

リストを指定する場合には、各ノードは、次の例に示すように、先頭にリテラル node_name を付けて行ごとに指定します。

```
node_name <node1>
node_name <node2>
node_name <node3>
```

ノード名を指定する順序は重要です。最初にプライマリノード名 (パッケージを通常起動するノード) を指定して、次に 1 番目の引き継ぎノード名 (フェイルオーバー先の第 1 候補)、2 番目の引き継ぎノード名、その他のノード名と、優先する順に指定します。

フェイルオーバーでは、パッケージの制御はパッケージ構成ファイルで指定されている次の引き継ぎノードへ移行されます。そのノードが利用できないか、その時点でパッケージを実行できない場合は、リスト上のその次のノードへと順に選択されます。

- ① **重要:** ノード名についての重要な情報は、「[クラスター構成のパラメーター](#)」(82 ページ) を参照してください。

クロスサブネットパッケージを構成する際の留意点については、「[クロスサブネットフェイルオーバーについて](#)」(118 ページ) を参照してください。クロスサブネットパッケージについては、「[クロスサブネット構成](#)」(25 ページ) で詳細に説明しています。

auto_run

yes または no を設定できます。デフォルトは、yes です。

フェイルオーバーパッケージの場合には、yes を設定すると、Serviceguard は、クラスターの起動時に (node_name にリストされている最初の利用可能なノードで) パッケージを起動し、ノードで障害が発生した場合には、引き継ぎノードでパッケージを自動的に再起動します。no を設定すると、Serviceguard は、パッケージの自動起動や、別のノードでの再起動を行いません。

フェイルオーバーはパッケージの切り替えと呼ばれることもあります。パッケージの切り替えは、パッケージの動作中に cmmmodpkg コマンドを使って有効/無効にすることができます。

パッケージが他のパッケージに依存しているか、他のパッケージの依存対象となっている場合は、auto_run に yes を設定してください。「[パッケージの依存関係について](#)」(100 ページ) を参照してください。

システムマルチノードパッケージの場合には、auto_run に yes を設定する必要があります。マルチノードパッケージでは、auto_run に yes を設定すると、クラスターに参加する新しいノード上でパッケージのインスタンスが起動されます。no の場合は、起動されません。

node_fail_fast_enabled

yes または no を設定できます。デフォルトは、no です。

yes は、パッケージで障害が発生した場合に、パッケージが動作しているノードが停止 (リブート) することを意味します。no の場合には、Serviceguard はシステムを停止しません。

このパラメーターに yes を設定しておくで、以下のいずれかの事象が発生した場合に、Serviceguard は、制御スクリプトで障害が発生したノードで、システムを停止 (リブート) します。

- パッケージのサブネットに障害が発生し、利用できるバックアップ用ネットワークがない

- 汎用リソースに障害が発生した
- Serviceguard が停止機能を実行できない
- 起動または停止機能がタイムアウトした

注記: パッケージ停止機能が "exit 1" で失敗した場合、Serviceguard はノードを停止させませんが、そのパッケージに `no_restart` を設定します。この設定によりパッケージ切り替えが無効になり、[auto_run\(164 ページ\)](#) が `no` に設定されるので、そのパッケージはどの引き継ぎノードでも開始されなくなります。

`node_fail_fast_enabled` に `yes` を設定すると、Serviceguard が同一ノードでパッケージの起動を複数回試みる (そして失敗する) ことが防止されます。

`node_fail_fast_enabled` を `yes` に設定すると、パッケージは正しく停止できない場合でも、別のノードにフェイルオーバーできます。`node_fail_fast_enabled` を使用すると、ノード上の**すべての**パッケージが突然停止するため、この設定を使用する場合は注意が必要です。詳細は、「障害への応答」(68 ページ) と「パッケージ障害とサービス障害への応答」(70 ページ) を参照してください。

システムマルチノードパッケージの場合には、`node_fail_fast_enabled` には `yes` を設定する必要があります。

`run_script_timeout`

パッケージの起動に許される時間 (秒単位)、または `no_timeout` です。デフォルトは、`no_timeout` です。最大値は、4294 です。

パッケージが `run_script_timeout` で指定された時間内に起動を完了できなかった場合には、Serviceguard はパッケージを終了させて、別のノードに切り替えられることがないようにします。このとき、`node_fail_fast_enabled` に `yes` が設定されていると、ノードは停止 (リブート) されます。

タイムアウトなしを指定した場合 (`no_timeout`) には、Serviceguard はパッケージが起動するのを無限に待ちます。

タイムアウトが発生すると、以下の処理が実行されます。

- 切り替えは無効になります。
- 現在のノードでパッケージを実行できなくなります。

注記: `no_timeout` を指定しており、検証ステップ (`cmcheckconf (1m)`) の間にスクリプトがハングした場合、あるいはスクリプトの完了に非常に長い時間を要する場合、`cmcheckconf` は、20 分待ってから検証の完了を断念します。

`halt_script_timeout`

パッケージの停止に許される時間 (秒単位)、または `no_timeout` です。デフォルトは、`no_timeout` です。最大値は、4294 です。

パッケージの停止プロセスが `halt_script_timeout` で指定された時間内に完了しなかった場合には、Serviceguard はパッケージを終了させて、別のノードに切り替えられることがないようにします。このとき、`node_fail_fast_enabled(164 ページ)` に `yes` が設定されていると、ノードは停止 (リブート) されます。

`halt_script_timeout` を指定する場合には、その値はこのパッケージの [service_halt_timeout\(172 ページ\)](#) に設定されているすべての値の合計よりも大きくなければなりません。

タイムアウトが発生すると、以下の処理が実行されます。

- 切り替えは無効になります。
- 現在のノードでパッケージを実行できなくなります。

停止スクリプトでタイムアウトが発生した場合には、手作業でクリーンアップする必要があります。第8章「[クラスターのトラブルシューティング](#)」(234 ページ) を参照してください。

`successor_halt_timeout`

Serviceguard が、このパッケージを停止する前に、このパッケージに依存しているパッケージが停止するのを待機する時間 (秒単位) を指定します。0～4294 の値、または `no_timeout` を指定できます。デフォルトは、`no_timeout` です。

- `no_timeout` は、依存しているパッケージが停止するのを Serviceguard が無限に待つことを意味します。
- 0 は、Serviceguard がこのパッケージを停止する前に、依存しているパッケージが停止するのを待たないことを意味します。

A.11.18 での新規パラメーターです (モジュラーパッケージおよび従来のパッケージ用)。「[パッケージの依存関係について](#)」(100 ページ) も参照してください。

`script_log_file`

パッケージログファイルの絶対パス名です。デフォルトは、`$SGRUN/log/<package_name>.log` です。(Serviceguard のパス名についての詳細は、「[Serviceguard のファイルの位置](#)」(122 ページ) を参照してください。) `log_level` も参照してください。

`operation_sequence`

パッケージの構成要素モジュールで定義されるスクリプトの起動順序を定義します。詳細は、パッケージ構成ファイルを参照してください。

このパラメーターは構成できません。構成ファイル内のエントリーを変更しないでください。モジュラーパッケージで新たに導入されました。

`log_level`

パッケージの検証時に `stdout` に出力される情報の量と、パッケージの起動/停止時に `script_log_file` に出力される情報の量を決定します。有効な値は0～5 ですが、通常は最初の 2 つ (0 または 1) だけを使用してください。残り (2～5) は、HP のサポートによって使用されます。

- 0: 情報メッセージ
- 1: 詳細情報が少し追加された情報メッセージ
- 2: ロジックフローを示すメッセージ
- 3: 詳細なデータ構造情報を示すメッセージ
- 4: 詳細なデバッグ情報
- 5: 関数呼び出しフロー

モジュラーパッケージで新たに導入されました。

`failover_policy`

パッケージ障害発生時に、パッケージを起動または再起動する場所を Serviceguard が決める際の判断基準を指定します。`configured_node` または `min_package_node` を設定できます。デフォルトは、`configured_node` です。

- `configured_node` は、Serviceguard が、[node_name](#) (164 ページ) で指定されているリスト内の最初に利用可能なノードで、パッケージの起動を試みることを意味します。
- `min_package_node` は、Serviceguard が、`node_name` リスト内のノードのうち、その時点で実行しているパッケージの数が最も少ないノードで、パッケージを起動することを意味します。

このパラメーターが設定できるのは、フェイルオーバーパッケージだけです。このパッケージが他のパッケージと依存関係がある場合には、「[パッケージの依存関係について](#)」(100 ページ)を参照してください。

failback_policy

一次ノード (*node_name* リスト内の最初のノード) が再び利用可能になったときに、一次ノードで実行されていないパッケージを Serviceguard が自動的に移動させるかどうかを指定します。automatic または manual を設定できます。デフォルトは、manual です。

- manual は、パッケージが現在のノードで引き続き実行を続けることを意味します。
- automatic は、プライマリノードが利用可能になり次第、Serviceguard がパッケージをプライマリノードに移動させることを意味します。ただし、移動させるためには、それよりも高い *priority* を持つパッケージを移動させなくてはならない場合には、移動させません。

△ **注意:** *failback_policy* が automatic のときに *NODE_NAME* を「*」に設定し、クラスター内のノードを追加、削除したり、またはその名前を変更したりすると、パッケージのプライマリノードが変更されて、自動的にパッケージのフェイルオーバーが実行されることがあります。

このパラメーターが設定できるのは、フェイルオーバーパッケージだけです。このパッケージが他のパッケージと依存関係がある場合には、「[パッケージの依存関係について](#)」(100 ページ)を参照してください。

priority

failover_policy に configured_node が設定されているフェイルオーバーパッケージに優先順位を割り当てます。有効な値は、1〜3000 または no_priority です。デフォルトは、no_priority です。dependency_ パラメーターの説明([167 ページ](#))も参照してください。

priority を使うと、パッケージの起動時やフェイルオーバーまたはフェイルバックが必要になったときに、依存関係を満たすことができます。依存対象のパッケージよりも高い優先順位を持つパッケージでは、選択したノード上で、依存対象のパッケージが強制的に起動または再起動され、依存関係が満たされます。

優先順位を割り当てる場合、クラスター内で一意である必要があります。小さい数値は高い優先順位を示します。数値で指定した優先順位は no_priority よりも高い優先順位を意味します。シーケンスに間隔を空けるために、値を 20 きざみで割り当てることをお勧めします。そうしないと、新しいパッケージに優先順位を割り当てるときに、既存の優先順位をすべて変更しなければならなくなることがあります。

① **重要:** 優先順位は順位付けであるため、数値が小さい方が高い優先度を表します (20 は 40 よりも優先順位が高くなります)。数値の優先順位は、no_priority よりも高い順位です。

A.11.18 での新規パラメーターです (モジュラーパッケージおよび従来のパッケージ用)。詳細は、「[パッケージの依存関係について](#)」(100 ページ)を参照してください。

dependency_name

このパッケージを実行させる (または実行を継続させる) ために満たす必要がある特定の依存関係を示す固有 ID です (*dependency_condition* を参照してください)。この ID は、このパッケージの *dependency_name* 内で一意でなければなりません。名前の長さや形式に対する制限は、*package_name* ([163 ページ](#)) の場合と同じです。

① **重要:** Serviceguard の以前のリリースでの依存関係名の制限は、これよりも緩いものでした。上記の規則に準拠していない *dependency_name* が指定されたパッケージでも実行できますが、再構成時には *dependency_name* を変更する必要があります。cmcheckconf と cmapplifyconf では、新しい規則が強制されます。

このパッケージが別のパッケージに依存している場合には、`dependency_condition` と `dependency_location`、およびオプションで `priority`(167 ページ) とともにこのパラメーターを設定します。たとえば、このパッケージが `pkg2` に依存している場合には、次のように設定します。

```
dependency_name pkg2dep
dependency_condition pkg2 = UP
dependency_location same_node
```

パッケージの依存関係についての詳細は、「[パッケージの依存関係について](#)」(100 ページ) を参照してください。

`dependency_condition`

この依存関係を満たすために、成立しなければならない条件です。Serviceguard A.11.18 から、設定できる条件は他のパッケージが動作していることだけになりました。

構文は、`<package_name> = UP` です。`<package_name>` には依存しているパッケージの名前を指定します。現在のパッケージ(構成中のパッケージ)のタイプと特性によって、依存対象のパッケージのタイプが以下のように制限されます。

- 現在のパッケージがマルチノードパッケージの場合には、`<package_name>` に指定できるのはマルチノードパッケージまたはシステムマルチノードパッケージのみ
- 現在のパッケージがフェイルオーバーパッケージで、その `failover_policy`(166 ページ) が `min_package_node` の場合には、`<package_name>` に指定できるのはマルチノードパッケージまたはシステムマルチノードパッケージのみ
- 現在のパッケージがフェイルオーバーパッケージで、`failover_policy` が `configured_node` の場合には、`<package_name>` に指定できるのはマルチノードパッケージ、システムマルチノードパッケージ、または `failover_policy` が `configured_node` のフェイルオーバーパッケージのみ

「[パッケージの依存関係について](#)」(100 ページ) も参照してください。

`dependency_location`

`dependency_condition` を満たすべき場所を指定します。有効な値は、`same_node` だけです。

`weight_name`、`weight_value`

これらのパラメーターは、パッケージの重みを指定します。この重みが、ノードに使用可能なキャパシティ(クラスター構成ファイル内で `CAPACITY_NAME` と `CAPACITY_VALUE` パラメーターで定義される)と比較され、パッケージをその場所で実行できるかどうか判断されます。

どちらのパラメーターも省略可能ですが、`weight_value` を指定する場合は `weight_name` も指定する必要があり、こちらのパラメーターを先に指定する必要があります。クラスターごとに、4つのキャパシティに対応する最大4つの重みを定義できます。このパッケージに複数の重みを指定するには、`weight_name` と `weight_value` を繰り返してください。

注記: ただし、`weight_name` が `package_limit` である場合、クラスター全体を通して1つの重みとキャパシティだけを使用できます。`package_limit` は予備の値であり、使用する場合にはその形式で正確に入力する必要があります。これは、重みとキャパシティを管理するのに最も簡単な方法です。詳細は、「[単純な方法](#)」(108 ページ) を参照してください。

`weight_name` の形成規則は `package_name`(163 ページ) の形成規則と同じです。`weight_name` は、対応する **CAPACITY_NAME** と完全に一致する必要があります。

`weight_value` は、0～1000000 の範囲の符号なし浮動小数点数です。小数点以下は最大3桁まで指定できます。

これらのパラメーターを使用すると、指定したノードのキャパシティに対応する、クラスター全体でのデフォルトのパッケージ重みを上書きすることができます。クラスター全体のデフォ

ルトパッケージ重みは、クラスター構成ファイル内の `WEIGHT_NAME` パラメーターと `WEIGHT_DEFAULT` パラメーターを使用して定義できます (明示的なデフォルト)。明示的なデフォルトを定義しないと (つまり、クラスター構成ファイルで対応する `WEIGHT_NAME` と `WEIGHT_DEFAULT` を指定せずに `CAPACITY_NAME` を定義すると)、デフォルト重みはゼロとみなされます (暗黙のデフォルト)。パッケージ構成ファイルで `weight_name` と `weight_value` を構成すると、クラスター全体でのデフォルト (暗黙的または明示的) が上書きされ、このパッケージに特定の重みが割り当てられます。

詳細は、「[パッケージの重みについて](#)」(108 ページ) を参照してください。「[クラスター構成のパラメーター](#)」(82 ページ)、`cmmakepkg` (1m) および `cmquerycl` (1m) のマンページ、クラスター構成とパッケージ構成のテンプレートファイルにある関連するパラメーターについての説明も参照してください。

`monitored_subnet`

このパッケージ用に監視する LAN サブネットです。従来の `SUBNET` の代わりとなるパラメーターです。このパラメーターは、従来のパッケージではパッケージ構成ファイルで引き続きサポートされています。「[従来のパッケージの構成](#)」(218 ページ) を参照してください。

複数のサブネットを指定できます。サブネットごとに行を分けて指定します。

サブネットを `monitored_subnet` として指定した場合、パッケージは、そのサブネットから到達できないノードでは実行されません。これは通常、サブネットが動作状態でなければパッケージは実行されないことを意味します。(サブネットをあるノードでは構成できて、他のノードでは構成できないようなクロスサブネット構成については、次の

[monitored_subnet_access](#)、[ip_subnet_node](#) (171 ページ)、および「[クロスサブネットフェイルオーバーについて](#)」(118 ページ) を参照してください)。

通常、このパラメーターと `ip_subnet` パラメーター(170 ページ) で指定することで、`ip_subnet` を監視しますが、他のサブネットを監視しなければならない場合もあります。クラスターに構成されている (クラスター構成ファイルの `STATIONARY_IP` パラメーターによって) 任意のサブネットを指定することができます。詳細は、「[定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット](#)」(54 ページ) を参照してください。

`monitored_subnet` のいずれかで障害が発生すると、Serviceguard は、`node_name` (164 ページ) で指定されているノードのうち、このパッケージ用に定義されているすべての `monitored_subnet` で通信できる他のノードにパッケージを切り替えます。詳細および例は、構成ファイル内のコメントを参照してください。

`monitored_subnet_access`

クロスサブネット構成では、各 `monitored_subnet` がパッケージの `node_name` リスト (164 ページ) のすべてのノード上でアクセス可能か、一部のノード上でのみアクセス可能かを指定します。有効な値は `PARTIAL` と `FULL` です。`PARTIAL` は、すべてのノードではないが、少なくとも 1 つのノードがサブネットにアクセスできます。`FULL` では、すべてのノードがサブネットにアクセスできます。デフォルトは `FULL` であり、`monitored_subnet_access` が指定されない場合にはこれが有効になります。

[ip_subnet_node](#) (171 ページ) と「[クロスサブネットフェイルオーバーについて](#)」(118 ページ) も参照してください。

モジュラーパッケージで新たに導入されました。従来のパッケージについては、「[クロスサブネットフェイルオーバーの構成](#)」(224 ページ) を参照してください。

`monitored_subnet_access`

クロスサブネット構成で、各 `monitored_subnet` に、パッケージのノードリスト ([node_name](#) (164 ページ) を参照) 中のすべてのノードがアクセス可能か、一部のノードのみがアクセス可能かを指定します。有効な値は `PARTIAL` と `FULL` です。`PARTIAL` は、すべてのノードではないが、少なくとも 1 つのノードがサブネットにアクセスできます。`FULL` では、すべてのノード

がサブネットにアクセスできます。デフォルトはFULLであり、`monitored_subnet_access` が指定されない場合にはこれが有効になります。

[ip_subnet_node](#) (171 ページ) と「クロスサブネットフェイルオーバーについて」 (118 ページ) も参照してください。

モジュラーパッケージで新たに導入されました。従来のパッケージについては、「クロスサブネットフェイルオーバーの構成」 (224 ページ) を参照してください。

`ip_subnet`

パッケージで使われる IP サブネットを指定します。`SUBNET` の代わりとなるパラメーターです。このパラメーターは、従来のパッケージではパッケージ制御スクリプトで引き続きサポートされています。

- △ **注意:** このサブネットはクラスターに構成することをお勧めします。これをクラスター構成ファイルで行うには、このパッケージの `NODE_NAME` リスト内の各ノードについて、同じサブネット上の `NETWORK_INTERFACE` の下で `HEARTBEAT_IP` または `STATIONARY_IP` を指定します。たとえば、クラスター構成ファイルで以下のエントリーを指定すると、ノード `ftsys9` 上にサブネット `192.10.25.0 (lan1)` が構成されます。

```
NODE_NAME ftsys9
NETWORK_INTERFACE lan1
HEARTBEAT_IP 192.10.25.18
```

詳細は、「[クラスター構成のパラメーター](#)」 (82 ページ) を参照してください。

クラスターにこのサブネットを構成しないと、Serviceguard はこのサブネットを管理することも監視することもできません。実際、パッケージの `node-name` リスト (164 ページ) に挙げられたすべてのノードでこのサブネットが利用できるという保証がありません。このようなサブネットは外部サブネットと呼ばれ、そのサブネット上の再配置可能アドレスは外部アドレスと呼ばれます。外部サブネットを使用すると、以下の状況が発生するリスクがあります。

- サブネットで障害が発生しても、パッケージが代替ノードにフェイルオーバーしない。
- サブネットで障害が発生しなくても、他のタイプの障害のためにパッケージのフェイルオーバーが必要となった場合、引き継ぎノード上でそのサブネットが利用できないために引き継ぎノードでの起動に失敗する可能性がある。

このため、すべての `ip_subnet` をクラスターに構成してください。ただし、DLPI をサポートしていないネットワークング技術を使用する場合は除きます。このようなケースでは、ネットワーク製品のマニュアルの指示に従って、その製品を Serviceguard に組み込んでください。

使用する各サブネットに対して、1 つの行に 1 つのサブネットアドレスを指定し、それに続く行に、そのサブネット上でそのパッケージが使う再配置可能 IP アドレスを指定します。これらのアドレスは、パッケージの起動時に構成され、停止時に構成解除されます。

たとえば、このパッケージでサブネット `192.10.25.0` と、再配置可能 IP アドレス `192.10.25.12` と `192.10.25.13` を使う場合には、次のように指定します。

```
ip_subnet 192.10.25.0
ip_address 192.10.25.12
ip_address 192.10.25.13
```

サブネットを監視する場合は、`monitored_subnet` パラメーター (169 ページ) でも指定してください。

クロスサブネット構成では、どのノードでサブネットが構成されるかを指定する必要もあります。次の `ip_subnet_node` を参照してください。[monitored_subnet_access](#) (169 ページ) と「クロスサブネットフェイルオーバーについて」 (118 ページ) も参照してください。

このパラメーターが設定できるのは、フェイルオーバーパッケージだけです。

ip_subnet_node

クロスサブネット構成では、`ip_subnet` がどのノードで構成されるかを指定します。`ip_subnet` の下に `ip_subnet_node` がリストされていない場合は、`'ip_subnet(164 ページ)` はこのパッケージの `node_name` リストにあるすべてのノードで構成されているものとみなされます。

パッケージの稼働中に追加や削除を行うことができますが、次の制限があります。

- 追加や削除を行うノードではパッケージが実行されていない。
- ノードは、この `ip_subnet` の `ip_subnet_node` リストで、追加する最初のノード、または削除する最後のノードであってはならない。

`monitored_subnet_access(169 ページ)` と「クロスサブネットフェイルオーバーについて」(118 ページ) も参照してください。

モジュラーパッケージで新たに導入されました。従来のパッケージについては、「クロスサブネットフェイルオーバーの構成」(224 ページ) を参照してください。

ip_address

指定した `ip_subnet` 上の再配置可能 IP アドレスです。IP の代わりとなるパラメーターです。このパラメーターは従来のパッケージではパッケージ制御スクリプトで引き続きサポートされています。

再配置可能 IP アドレスについての詳細は、「定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット」(54 ページ) を参照してください。

このパラメーターが設定できるのは、フェイルオーバーパッケージだけです。

service_name

サービスは、パッケージが動作している限り Serviceguard によって監視されるプログラムまたは機能です。`service_name` にはこの機能を指定し、`cmrunserv` コマンドと `cmhaltserv` コマンドで使われます。1 つのパッケージには最大で 30 サービス、1 つのクラスターには最大で 900 サービスを構成できます。

名前の長さや形式に関する制限は、`package_name(163 ページ)` に対するものと同じです。`service_name` はクラスター内の全パッケージで一意である必要があります。

- ❶ **重要:** Serviceguard の以前のリリースでのパッケージ名の制限は、これよりも緩いものでした。上記の規則に準拠していない名前を持つサービスが指定されたパッケージでも実行できますが、再構成時には名前を変更する必要があります。`cmcheckconf` と `cmapplyconf` では、新しい規則が強制されます。

各サービスは、5 つのパラメーター `service_name`、`service_cmd`、`service_restart`、`service_fail_fast_enabled`、`service_halt_timeout` を使って定義します。以下の説明を参照してください。

以下に、完全に定義されたサービスの例を示します。

```
service_name      patricks-package4-ping]
service_cmd       "/usr/sbin/ping hasupt22"
service_restart   unlimited
service_fail_fast_enabled no
service_halt_timeout      300
```

その他の例については、パッケージ構成テンプレートファイルを参照してください。

従来のパッケージでは、このパラメーターはパッケージ制御スクリプトとパッケージ構成ファイルに記述されていました。

service_cmd

この `service_name` に対応するプログラムまたは関数を起動するコマンドです。次に例を示します。

```
/usr/bin/X11/xclock -display 15.244.58.208:0
```

絶対パス名で指定する必要があります。PATH変数などの環境変数はコマンドに渡されません。デフォルトのシェルは、/bin/sh です。

注記: サービス起動コマンドの定義は、慎重に行ってください。各起動コマンドは、以下のように実行されます。

- cmrunserv コマンドは、起動コマンドを実行します。
- Serviceguard は、起動コマンドが生成したプロセスのプロセス ID (PID) を監視します。
- コマンドが終了すると、Serviceguard は、障害が発生したと判断し、パッケージを引き継ぎノードに移動するなど、適切な対処を実行します。
- 起動コマンドが、他のコマンドを起動して終了するシェルスクリプトの場合、Serviceguard はこの正常終了を障害とみなします。

各起動コマンドには実際のサービス名を指定し、実際のサービスが停止するまでプロセスが終了しないようにしてください。このためには、サービスが監視プログラムとなるようパッケージを構成することも1つの方法です。パッケージの主要な機能を構成するアプリケーションの状態をチェックし、アプリケーションの障害を認識した場合に終了するようなプログラムにします。アプリケーション自体は、[external_script \(178 ページ\)](#) によって起動できます。

このパラメーターは、従来のパッケージではパッケージ制御スクリプトにあります。

`service_restart`

Serviceguard が、`service_cmd` の再起動を試みる回数です。有効な値は、unlimited、none、または任意の正の整数です。デフォルトは、none です。

値にunlimitedを指定すると、サービスは無限に再起動されます。値にnoneを指定すると、サービスは再起動されません。

このパラメーターは、従来のパッケージではパッケージ制御スクリプトにあります。

`service_fail_fast_enabled`

`service_name` で指定したサービスが失敗した場合に、パッケージが動作しているノードをServiceguard が停止 (リブート) するかどうかを指定します。有効な値は、yes とno です。デフォルトは、no です。これはこのサービスが失敗してもノードを停止させないことを意味します。

`service_halt_timeout`

Serviceguard が、サービスのプロセスを強制的に終了させる前にサービスが停止するのを待つ時間 (秒単位) です。最大値は、4294 です。

この値は、サービスで必要なクリーンアップが完了するのに十分な時間である必要があります。

値を指定しないと、タイムアウト値はゼロとみなされます。これは、Serviceguard がまったく待機せずにプロセスを停止することを意味します。

`generic_resource_name`

パッケージ内の汎用リソースを識別するために使用する論理名を定義します。この名前は、`cmgetresource (1m)` および `cmsetresource (1m)` コマンドで使用される汎用リソース名に対応します。

1 つのパッケージで複数の `generic_resource_name` エントリーを指定できます。

名前の長さや形式に対する制限は、[package_name \(163 ページ\)](#) の場合と同じです。

それぞれの名前がパッケージ内部で一意でなければなりませんが、1 つのリソースを複数のパッケージを対象に指定できます。

1 つのクラスターで最大 100 個の汎用リソースを構成できます。
各汎用リソースは次の 3 つのパラメーターで定義されます。

- `generic_resource_name`
- `generic_resource_evaluation_type`
- `generic_resource_up_criteria`

以下の説明を参照してください。

以下に、汎用リソースパラメーターの定義の例を示します。

```
generic_resource_name      cpu_monitor
generic_resource_evaluation_type  during_package_start
generic_resource_up_criteria    <50
```

その他の例については、パッケージ構成ファイルを参照してください。

`generic_resource_evaluation_type`

汎用リソースのステータスの評価を行うタイミングを定義します。

有効な値は、`during_package_start` および `before_package_start` です。デフォルトの値は、`during_package_start` です。

パッケージの一連の起動プロセス最中に利用できるようになるリソースは、`evaluation_type` を `during_package_start` に指定して構成する必要があります。

このような汎用リソースの監視は、パッケージの一部として開始および停止でき、監視スクリプトはサービスとして構成できます。監視用実行可能ファイルとスクリプトのフルパス名を含む `service_name` と `service_cmd` を構成することで実現できます。汎用リソースの監視が開始されるのは、監視スクリプトの起動時だけです。パッケージの起動時に開始されるわけではありません。

監視スクリプトの詳細は、「[汎用リソース用の監視スクリプト](#)」(279 ページ)を参照してください。

複数のパッケージの一部として監視される必要のある共通の汎用リソースがある場合は、そのリソースの監視スクリプトを 1 つのパッケージの一部として起動できるため、他のすべてのパッケージが同じ監視スクリプトを使用できます。共通のリソースに対して複数のモニターを起動する必要はありません。監視スクリプトを起動したパッケージで障害が発生するかまたはパッケージが停止すると、この共通リソースを使っている他のすべてのパッケージも停止します。

これらのリソースには、通常、`evaluation_type` の `before_package_start` が指定され、マルチノードパッケージで監視スクリプトを構成することが推奨されます。

パッケージが起動するにはこれらのリソースを使用できなければなりません (ステータスが「up」)。また、これらのリソースの監視スクリプトは、アプリケーションパッケージの外部で構成する必要があります。

`generic_resource_up_criteria`

`generic_resource_name` で識別される汎用リソースのステータスが「up」かどうかを判断する基準を定義します。

属性として、論理演算子と値が必要です。演算子として使用できるのは、`==`、`!=`、`>`、`<`、`>=`、および `<=` です。値は、1～2147483647 の正の整数でなければなりません。

注記: 上記以外の演算子はサポートされていません。この属性は、複数の `up_criteria` は受け付けません。たとえば、`>> 10`、または `<< 100` は有効ではありません。

値は上記の演算子と組み合わせて 1～2147483647 の範囲で入力できますが、次に示す 4 つの条件の設定は許可されていません。

`< 1`、`> 2147483647`、`>= 1`、および `<= 2147483647`

以下では、その理由を説明します。

- `generic_resource_up_criteria` を `< 1` または `> 2147483647` に指定すると、リソースのステータスを「up」にする `up_criteria` の条件を満たす値を入力できなくなります。このため、リソースは永遠に「up」になれません。
 - 同様に、`generic_resource_up_criteria` を `>= 1` または `<= 2147483647` に指定すると、条件が必ず満たされるためステータスは常に「up」になります。`up_criteria` の条件を満たさない値を入力できなくなり、リソースのステータスを「down」にすることができません。
-
- `generic_resource_up_criteria` は、オプションの属性です。この属性は、ある汎用リソースがシンプル汎用リソースかそれとも拡張汎用リソースかを決定します。この属性は、シンプルリソースについては指定されませんが、拡張リソースでは必要です。
 - あるパッケージに、シンプルリソースと拡張リソースの両方を含むことができます。
 - 1 つのパッケージでシンプル汎用リソースとして構成したリソースを、別のパッケージで拡張汎用リソースとして設定することはできません。リソースは、すべてのパッケージでシンプルリソースまたは拡張リソースのいずれかでなければなりません。
 - 1 つのパッケージで、`evaluation_type` が `before_package_start` の汎用リソースと `during_package_start` の汎用リソースを組み合わせることができます。

`vgchange_cmd`

`VGCHANGE` の代わりとなるパラメーターです。このパラメーターは、従来のパッケージでは引き続きサポートされています。「従来のパッケージの構成」(218 ページ) を参照してください。`vg` エントリーで指定される各 Logical Volume Manager (LVM) ボリュームグループのアクティブ化方法を指定します。

デフォルトは、`vgchange -a y` です。

`vg`

ファイルシステム (Red Hat GFS 以外。 `fs_type` を参照) をマウントする必要がある LVM ボリュームグループを指定します (1 行に 1 つの `vg` を指定します)。対応する `vgchange_cmd` (上記参照) には、ボリュームグループのアクティブ化方法を指定します。パッケージスクリプトは、`fs` パラメーター (「ファイルシステムパラメーター」を参照) に基づいて、必要なファイルシステムコマンドを生成します。

ファイルシステムパラメーター

パッケージは起動時に 1 つ以上のストレージグループをアクティブ化し、ファイルシステムの論理ボリュームをマウントすることができます。停止時に、パッケージスクリプトはファイルシステムをマウント解除し、各ストレージグループを非アクティブ化します。すべてのストレージグループは、各ターゲットノード上でアクセス可能でなければなりません。

パッケージ構成ファイルで指定した各ファイルシステム (`fs_name`) に対して、論理ボリューム、マウントポイント、`mount`、`umount`、`fsck` の各オプション、およびファイルシステムのタイプを指定しなければなりません。例を次に示します。

`fs_name /dev/vg01/lvol1`

```
fs_directory /pkg01aa
fs_mount_opt "-o rw"
fs_umount_opt ""
fs_fsck_opt ""
fs_type "ext3"
```

論理ボリュームは、LVM ボリュームグループ上に構築されている必要があります。論理ボリュームはどのような順番でも入力できます。

gfs ファイルシステムを構成するには、パラメーター `fs_name`、`fs_directory`、および `fs_mount_opt` だけを使います。例については構成ファイルを参照してください。gfs については、`fs_type` で説明しているルールも適用されます。

注記: Red Hat GFS は Serviceguard A.11.20.00 ではサポートされません。

NFS インポートファイルシステムについては、`fs_name` (176 ページ) および `fs_server` (176 ページ) の説明を参照してください。

詳細については、以降のパラメーターの説明で示します。

`concurrent_fsck_operations`

パッケージの起動時にマウントされるファイルシステムで許される `fsck` の同時実行数です。Red Hat GFS (`fs_type` を参照) では使われません。

有効な値は、任意の正の数です。デフォルトは 1 です。

パッケージで多数のファイルシステムに対して `fsck` を実行する必要がある場合には、テスト段階でこのパラメーターを慎重にチューニングすることによって、性能を改善できます (値を少しずつ増やし、増やすたびに性能を監視します)。

`fs_mount_retry_count`

各ファイルシステムについての、`mount` のリトライ回数です。有効な値は、0 以上の整数です。デフォルトは 0 です。Red Hat GFS (`fs_type` を参照) で有効な値は 0 だけです。Red Hat GFS は Serviceguard A.11.20.00 ではサポートされません。

パッケージの起動時にマウントポイントがビジーで、`fs_mount_retry_count` に 0 が設定されている場合には、パッケージの起動は失敗します。

マウントポイントがビジーで、`fs_mount_retry_count` に 0 より大きな値が設定されている場合は、起動スクリプトはマウントポイントがビジーになる原因となっているユーザープロセスの抹消 (`fuser -ku`) を試みて、その後ファイルシステムのマウントを再度試みます。起動スクリプトは、この操作を `fs_mount_retry_count` で指定されている回数だけ試みます。

`fs_mount_retry_count` で指定されている回数だけマウント操作を試みても失敗する場合には、パッケージの起動は失敗となります。

このパラメーターは、従来のパッケージではパッケージ制御スクリプトにあります。

`fs_umount_retry_count`

各ファイルシステムについての、`umount` のリトライ回数です。`FS_UMOUNT_COUNT` の代わりとなるパラメーターです。このパラメーターは、従来のパッケージではパッケージ制御スクリプトで引き続きサポートされています。「従来のパッケージの構成」(218 ページ) を参照してください。

有効な値は、1 または正の数 (Red Hat GFS 以外のファイルシステムの場合) です。デフォルトは 1 です。`fs_mount_retry_count` と同様に動作します。

fs_name

このパラメーターは、*fs_directory*、*fs_type*、*fs_mount_opt*、*fs_umount_opt*、*fs_fsck_opt* とともに使われて、パッケージによってマウントされるファイルシステムを指定します。*LV* の代わりとなるパラメーターです。このパラメーターは、従来のパッケージではパッケージ制御スクリプトで引き続きサポートされています。

fs_name には、論理ボリュームのブロック型デバイスファイルを指定する必要があります。

NFS インポートファイルシステムでは、*fs_name* 以外に *fs_server*、*fs_directory*、*fs_type*、および *fs_mount_opt* パラメーターが必要です。例については、[fs_server \(176 ページ\)](#) を参照してください。

- △ **注意:** NFS インポートファイルシステムを構成してパッケージに組み込む前に、「[NFS マウントされたファイルシステムのプランニング](#)」(94 ページ)にある規則とガイドラインをよく読んで理解してください。また、「[クラスター構成のパラメーター](#)」(82 ページ)で説明するクラスターパラメーター *CONFIGURED_IO_TIMEOUT_EXTENSION* の設定を済ませてください。

ファイルシステムはパッケージ構成ファイルで指定された順番でマウントされ、逆の順番でマウント解除されます。

詳細と例については、「[ファイルシステムパラメーター](#)」(174 ページ)と構成ファイルの *FILESYSTEMS* セクションのコメントを参照してください。また、「[ボリュームマネージャーのプランニング](#)」(78 ページ)と、*mount* のマンページも参照してください。

注記: Red Hat GFS (*fs_type* を参照) 以外のファイルシステムでは、*fs_name* エントリで指定した各論理ボリュームのボリュームグループは、このファイルで定義する必要があります (*vg* (174 ページ) を使用)。

fs_server

NFS インポートファイルシステムで使う NFS サーバーの名前または IP アドレス (IPv4 または IPv6)。この場合、HPUX では *fs_type* を *nfs* に、*fs_mount_opt* を *-o llock* に設定し、Linux では *local_lock = all* を設定します。*fs_name* は *fs_server* からインポートされるディレクトリを指定し、*fs_directory* はローカルマウントポイントを指定します。

例:

```
fs_name           /var/opt/nfs/share1
fs_server         wagon
fs_directory      /nfs/mnt/share1
fs_type           nfs
#fs_mount_opt     local_lock =all
#fs_umount_opt
#fs_fsck_opt
```

注記: *fs_umount_opt* はオプションです。また、*fs_fsck_opt* は、NFS インポートファイルシステムでは使用しません。(この例ではどちらもコメントアウトしています。)

fs_mount_opt は Linux ではサポートされません。

fs_directory

fs_name で指定したファイルシステムのルートです。*FS* の代わりとなるパラメーターです。このパラメーターは、従来のパッケージではパッケージ制御スクリプトで引き続きサポートされています。「[従来のパッケージの構成](#)」(218 ページ)を参照してください。

詳細は、*mount* のマンページと構成ファイル内のコメントを参照してください。

fs_type

fs_name で指定したファイルシステムのタイプです。このパラメーターは、従来のパッケージではパッケージ制御スクリプトにあります。

NFS インポートファイルシステムについては、このパラメーターを `nfs` に設定する必要があります。「[fs_server](#)」(176 ページ)にある例を参照してください。

サポートされるタイプは、`ext3`、`ext4` (RHEL 6以降で)、`reiserfs`、および `gfs` です。
`Red Hat GFS` と `reiserfs` は、`Serviceguard A.11.20.00` バージョンではサポートされません。



警告! `ext4` ファイルシステムは、遅延アロケーションのメカニズムを用いています。このため、ファイルをディスクに書き込む動作は `ext3` とは異なります。`ext3` と違い、`ext4` ファイルシステムはトランザクションのコミット時にデータをディスクに書き込みません。つまり、データがディスクに書き込まれるまでの時間はより長くなります。プログラムで、データがディスクに書き込まれることを保証するには、`fsync()` など、データの整合性を保証する呼び出しを使用する必要があります。

注記: `gfs` (Red Hat Global File System (GFS)) を使ったパッケージでは、種類の異なる他のファイルシステムを使うことはできません。`vg` と `vgchange_cmd` (174 ページ) は、GFS ファイルシステムでは無効です。`Serviceguard` で GFS を使用方法の詳細は、『Clustering Linux Servers with the Concurrent Deployment of HP Serviceguard for Linux and Red Hat Global File Systems for RHEL5』を参照してください。これは <http://www.hp.com/go/linux-serviceguard-docs> -> [White papers] にあります。

また、`concurrent_fsck_operations` (175 ページ)、`fs_mount_retry_count`、`fs_umount_retry_count` (175 ページ)、および `fs_fsck_opt` (177 ページ) も参照してください。

詳細は、パッケージ構成ファイルテンプレート内のコメントを参照してください。

`fs_mount_opt`

`fs_name` で指定したファイルシステムの `mount` オプションです。詳細は、構成ファイル内のコメントを参照してください。このパラメーターは、従来のパッケージではパッケージ制御スクリプトにあります。

`fs_umount_opt`

`fs_name` で指定したファイルシステムの `umount` オプションです。詳細は、構成ファイル内のコメントを参照してください。このパラメーターは、従来のパッケージではパッケージ制御スクリプトにあります。

`fs_fsck_opt`

`fs_name` で指定したファイルシステムの `fsck` オプションです。`Red Hat GFS` には使用しません (`Red Hat GFS` は `Serviceguard A.11.20.00` ではサポートされません) (`fs_type` を参照してください)。このパラメーターは、従来のパッケージではパッケージ制御スクリプトにあります。

詳細は、`fsck` のマンページと、構成ファイル内のコメントを参照してください。

`pv`

Persistent Reservation (PR) が行われる対象となる物理ボリュームです (デバイスが PR をサポートしている場合)。



重要: このパラメーターは HP パートナーが専用使用するものです。HP パートナーはパッケージ構成ファイル内の指示に従う必要があります。

`Serviceguard` による PR の実装についての詳細は、「[Persistent Reservation について](#)」(65 ページ)を参照してください。

pev_

`cmgetpkgenv` コマンドを使って、`external_pre_script`、`external_script`、あるいはその両方に渡すことができるパッケージ環境変数を指定します。モジュラーパッケージで新たに導入されました。

変数名は、`pev_<variable_name>` の形式であり、英数字とアンダースコアしか使用できません。文字列 `pev` (小文字でも大文字でも可) の後に、アンダースコア (`_`) を続ける必要があります。

変数名と値には、最大で `MAXPATHLEN` 文字 (Linux システムでは 4096) の文字列を指定できます。

複数の変数を定義できます。詳細は、「[外部スクリプトについて](#)」(115 ページ) と、構成ファイル内のコメントを参照してください。

external_pre_script

パッケージの起動時にボリュームグループとディスクグループをアクティブ化する前と、パッケージの停止時にそれらを非アクティブ化した後に (すなわち、実質的には、パッケージ起動時の最初のステップとパッケージ停止時の最後のステップ)、実行される外部スクリプトの絶対パス名です。モジュラーパッケージで新たに導入されました。

複数の `external_pre_script` を指定した場合には、パッケージの起動時に、スクリプトはパッケージ構成ファイルに記述されている順番に実行されます。パッケージの停止時には、その逆の順番に実行されます。

詳細と例については、「[外部スクリプトについて](#)」(115 ページ) と、構成ファイル内のコメントを参照してください。

external_script

外部スクリプトの絶対パス名です。このスクリプトは、パッケージの主要な機能を構成するアプリケーションを起動/停止する手段として使われます。モジュラーパッケージで新たに導入されました。

このスクリプトは、パッケージの起動時に、ボリュームグループとファイルシステムをアクティブ化して IP アドレスを割り当てた後、サービスを起動する前に実行されます。また、パッケージの停止時には、サービスを停止した後、IP アドレスの削除やボリュームグループとファイルシステムの非アクティブ化を行う前に実行されます。

複数の `external_script` を指定した場合には、パッケージの起動時に、スクリプトはこのファイルに記述されている順番に実行されます。パッケージの停止時には、その逆の順番に実行されます。

詳細と例については、「[外部スクリプトについて](#)」(115 ページ) と、構成ファイル内のコメントを参照してください。 `service_cmd` (171 ページ) も参照してください。

user_host

`user_name` (178 ページ) で指定するユーザーが、パッケージ管理コマンドを実行できるシステムです。

有効な値は、`any_serviceguard_node`、`cluster_member_node`、または特定のクラスターノードです。特定のノードを指定する場合、正式なホスト名 (完全修飾されたドメイン名の `hostname` 部分と、**hostname** 部分のみ) である必要があります。 `user_name` の場合と同様に、キーワードを正確に指定するように注意してください。

user_name

このパッケージを管理する権限を持ったユーザーの名前を指定します。 `user_host` (178 ページ) と `user_role` も参照してください。これらの 3 つのパラメーターで、このパッケージのアクセス制御ポリシーが定義されます (「[クラスターへのアクセスの制御](#)」(146 ページ) を参

照)。これらのパラメーターは、`user_name`、`user_host`、`user_role` の順番で定義する必要があります。

`user_name` の有効な値は、`any_user`、または `user_host` の `/etc/passwd` に登録されているログイン名のうち最大で 8 つのログイン名です。

注記: `any_user` のスペルは正確に記述してください。少しでも違っていると、Serviceguard はユーザー名と解釈します。

パッケージ構成ファイル内で許可できる `user_role` は、そのパッケージに対する `package_admin` だけです。クラスター構成ファイルでは他のロールも許可できます。詳細と例については、「[アクセス制御ポリシーの設定](#)」(148 ページ) を参照してください。

`user_role`

`package_admin` を指定し、ユーザーが `cmrunpkg` コマンド、`cmhaltpkg` コマンド、`cmmodpkg` コマンド (および Serviceguard Manager の同等の機能) とクラスターの `monitor` ロールにアクセスできるようにする必要があります。詳細は、「[クラスターへのアクセスの制御](#)」(146 ページ) を参照してください。

従来のパッケージでのみ使われるその他のパラメーター

- ❶ **重要:** 以下のパラメーターは、従来のパッケージでのみ使われます。これらのパラメーターは、モジュラーパッケージでは使わないでください。詳細は、「[従来のパッケージ構成の作成](#)」(218 ページ) を参照してください。

`PATH`

スクリプトで使用するパスを指定します。

`SUBNET`

パッケージで監視される IP サブネットを指定します。

`RUN_SCRIPT` および `HALT_SCRIPT`

各スクリプトの絶対パス名を指定します。

これらの 2 つのパラメーターを使うと、必要に応じて従来のパッケージの実行命令と停止命令を別々のスクリプトに分けることができます。この場合、両方のスクリプトに同じ構成情報 (ノード名や IP アドレスなど) を含めるようにしてください。

ただし、ほとんどの場合、実行命令と停止命令の両方で同じスクリプトを使うことをお勧めします。(パッケージの起動時にはスクリプトにパラメーター `start` が渡され、停止時にはパラメーター `stop` が渡されます。)

`LV`

パッケージによってマウントされるファイルシステムが格納された論理ボリュームの名前です。

`FS`

パッケージによってマウントされるファイルシステムのマウントポイントの名前です。

`VGCHANGE`

[vgchange_cmd](#) (174 ページ) で説明しています。

パッケージ構成ファイルの生成

パッケージに必要な構成モジュールを選択 ([「パッケージモジュールの選択」](#) (157 ページ) を参照) したら、それらのモジュールを含むパッケージ構成ファイルを生成する準備ができたことになります。このパッケージ構成ファイルは、ベースモジュール (フェイルオーバーモジュール、マルチノードモジュール、システムマルチノードモジュール) と、ユーザーが追加することを決めたパラメーターを含むモジュールで構成されます。

準備作業

パッケージの構築を開始する前に、\$SGCONF ディレクトリ内に、パッケージ用のサブディレクトリを作成します。たとえば、次のように入力します。

```
mkdir $SGCONF/pkg1
```

(Serviceguard のパス名については、「[Serviceguard のファイルの位置](#)」(122 ページ)を参照してください。)

cmmakepkg の例

cmmakepkg コマンドは、パッケージ構成ファイルを生成します。以下にいくつかの例を示します。詳細は、cmmakepkg (1m) のマンページを参照してください。すべての例で、\$SGCONF/pkg1 ディレクトリ内に編集可能な構成ファイル pkg1.conf を作成します。

注記: cmmakepkg コマンド行でベースモジュール (つまり default や all) を指定しなかった場合には、cmmakepkg コマンドは、ユーザーが指定したモジュールを無視して、代わりにすべてのパラメーターを含むデフォルトの構成ファイルを生成します。

複雑なパッケージの場合、または設定すべきパラメーターが明確になっていない場合には、デフォルトが最善の選択です。下記の最初の例を参照してください。

cmmakepkg コマンドに -v オプションを指定して、オンラインで表示する情報の量や、構成ファイルに含める情報の量を制御することができます。有効な値は、0、1 および 2 です。-v 0 を指定すると、すべてのコメントが削除されます。-v 1 を指定すると、各パラメーターの簡単な見出しが表示されます。-v 2 を指定すると、各パラメーターの詳細説明が表示されます。デフォルトは、レベル 2 です。

- すべてのオプションモジュールを含む構成ファイルを生成します。

```
cmmakepkg $SGCONF/pkg1/pkg1.conf
```

- 汎用的なフェイルオーバーパッケージを作成します (編集しないで適用できます)。

```
cmmakepkg -n pkg1 -m sg/failover $SGCONF/pkg1/pkg1.conf
```

- 再配置可能 IP アドレスを使い、実行時にファイルシステムがマウントされている必要があるアプリケーションを実行するフェイルオーバーパッケージ用の構成ファイルを生成します (コマンド全体を 1 行で入力します)。

```
cmmakepkg -m sg/failover -m sg/package_ip -m sg/service -m  
sg/filesystem -m sg/volume_group $SGCONF/pkg1/pkg1.conf
```

- 既存のパッケージに 汎用リソースモジュールを追加する構成ファイルを生成するには、以下のように指定します (コマンド全体を 1 行で入力します)。

```
cmmakepkg -i $SGCONF/pkg1/pkg1.conf -m sg/generic_resource
```

- 別のパッケージが動作中であることを必要とするアプリケーションを実行するフェイルオーバーパッケージ用の構成ファイルを生成します (コマンド全体を 1 行で入力します)。

```
cmmakepkg -m sg/failover -m sg/dependency -m sg/service  
$SGCONF/pkg1/pkg1.conf
```

- 既存のパッケージに services モジュールを追加する構成ファイルを生成するには、以下のように指定します (コマンド全体を 1 行で入力します)。

```
cmmakepkg -i $SGCONF/pkg1/pkg1.conf -m sg/service  
$SGCONF/pkg1/pkg1_v2.conf
```

注記: 一度に複数のモジュールを追加できます。

- 既存のパッケージに Persistent Reservation モジュールを追加する構成ファイルを生成するには、以下のように指定します。

```
cmmakepkg -i $SGCONF/pkg1/pkg1.conf -m sg/pr_cntl
```

次の手順

次の手順は、生成した構成ファイルを編集することです。「構成ファイルの編集」(181 ページ)を参照してください。

構成ファイルの編集

パッケージに必要なモジュールを含む構成ファイルを生成(「パッケージ構成ファイルの生成」(179 ページ)を参照)したら、ファイルを編集して、パッケージが意図したとおりに機能するようにパッケージのパラメーターに値を設定する必要があります。

複雑なフェイルオーバーパッケージは、以下の手順で段階的に構成することをお勧めします。

1. ボリュームグループとマウントポイントだけを構成します。
2. 構成を確認して適用します(「パッケージ構成の検証と適用」(184 ページ)を参照)。
3. パッケージを起動して、ノード間でパッケージが移動できることを確かめます。

注記: `cmcheckconf` と `cmapplyconf` は、マウントポイントやボリュームグループが存在するかチェックします。

4. パッケージを停止します。
5. パッケージの IP アドレスとアプリケーションのサービスを構成します。
6. パッケージを起動して、アプリケーションが正しく動作することと、サービス中断時のパッケージのフェイルオーバーが正しく行われることを確かめます。「パッケージマネージャーのテスト」(234 ページ)を参照してください。

以下の個条書きの項目をチェックリストとして使います。各パラメーターの詳細は、「パッケージのパラメーターの説明」(162 ページ)と構成ファイル内のコメントを参考にしてください。

注記: オプションのパラメーターは、構成ファイル内でコメントアウトされています(行の先頭に# が付けられています)。場合によっては、これらのパラメーターにはデフォルト値が設定されており、パラメーターのコメントを外し(# を削除し)、デフォルトと異なる有効な値を設定しない限り、このデフォルト値が使われます。ファイル内のコメントとこの章の説明を読んで、デフォルトを受け入れることと変更することの意味を理解してください。

いずれにしても、使うことに決めたパラメーターのコメントを外し、必要な値を設定する際には十分に注意してください。

-
- `package_name`: このパッケージの一意の名前を指定します。A.11.18 から、名前の形式が厳密に規定されたことに注意してください。
 - `package_type`: `failover` または `multi_node` を指定します(`system_multi_node` は、当社が提供する特殊用途のパッケージ用に予約されています)。他のパッケージがこのパッケージに依存している場合には、制限があります。「パッケージの依存関係について」(100 ページ)を参照してください。
詳細は、「パッケージのタイプ: フェイルオーバー、マルチノード、システムマルチノード」(158 ページ)を参照してください。

- `node_name`: このパッケージを実行できる各クラスターノードの名前を指定します。各ノードは、1 行に 1 つずつ指定します。
- `auto_run`: フェイルオーバーパッケージの場合に、`yes` を指定して、`node_name` で指定したノードのうち最初に利用可能なノードでパッケージを起動したり、ノードが故障した場合に、自動的にパッケージを再起動することを Serviceguard に許可します。`no` を指定すると、Serviceguard はパッケージを自動的に起動しなくなります。

- `node_fail_fast_enabled:yes` を指定すると、パッケージが失敗した場合に、ノードを停止 (システム停止) させることができます。それ以外の場合には、`no` を指定します。
- `run_script_timeout` と `halt_script_timeout`: それぞれ、Serviceguard がパッケージの起動または停止の完了を待つ時間 (秒数) を指定します。またはデフォルト `no_timeout` のままにしておきます。(165 ページ) を参照してください。
- `successor_halt_timeout`: 他のパッケージがこのパッケージに依存している場合に使います。「パッケージの依存関係について」(100 ページ) を参照してください。
- `script_log_file`(166 ページ)。
- `log_level`(166 ページ)。
- `failover_policy`(166 ページ)。 `configured_node` または `min_package_node` を指定します。

(このパラメーターが設定できるのは、フェイルオーバーパッケージだけです。)

- `failback_policy`(167 ページ)。 `automatic` または `manual` を指定します。
(このパラメーターが設定できるのは、フェイルオーバーパッケージだけです。)
- このパッケージが他のパッケージに依存する場合には、`dependency_name`、`dependency_condition`、`dependency_location`、そしてオプションで `priority` に値を設定します。

詳細は、「パッケージの依存関係について」(100 ページ) を参照してください。

注記: このパッケージが依存するパッケージは、このパッケージの検証時 (`cmcheckconf` を使用、「パッケージ構成の検証と適用」(184 ページ) を参照) にはクラスター構成にすでに含まれている必要があります。含まれていないと、検証は失敗します。

- パッケージの重みの構成には、`weight_name` および `weight_value` パラメーター(168 ページ) を使ってください。詳細は、「パッケージの重みについて」(108 ページ) を参照してください。
- `monitored_subnet` を使って、このパッケージ用に監視するサブネットを指定します。複数のサブネットがある場合には、1 行に 1 つずつ指定してこのパラメーターを必要なだけ繰り返します。
クロスサブネット構成では、必要に応じて、各 `monitored_subnet` に対し追加の `monitored_subnet_access` パラメーターを構成します。詳細は、「クロスサブネットフェイルオーバーについて」(118 ページ) を参照してください。
- パッケージで再配置可能 IP アドレスを使う場合には、`ip_subnet` アドレスと `ip_address` アドレスを入力します。規則と制限については、パラメーターの説明(170 ページ) を参照してください。
クロスサブネット構成では、必要に応じて、各 `ip_subnet` に対し追加の `ip_subnet_node` パラメーターを構成します。詳細は、「クロスサブネットフェイルオーバーについて」(118 ページ) を参照してください。
- パッケージが実行する各サービスについて、以下のパラメーターに値を設定します。
 - `service_name` を指定します (たとえば、デーモンまたは長時間実行するプロセス)。
 - `service_cmd` を指定します (たとえば、プロセスを起動するコマンド)。
 - `service_fail_fast_enabled` と `service_halt_timeout` に値を指定します (デフォルトから変更する必要がある場合)。
 - `service_restart` (パッケージのサービスが終了した場合にサービスを再起動したい場合。サービスの実行を繰り返す場合は、パッケージを終了して停止させるよりは、`unlimited` という値を指定すると便利です。)

パッケージが監視対象のディスクに依存する場合は、ディスク監視のためのサービスエントリを追加します。次のようなエントリを指定します。

```
service_name="cmresserviced_pkg1"
service_cmd="$SGBIN/cmresserviced /dev/sdd1"
service_restart=""
```

詳細は、「[ディスク監視構成の作成](#)」(184 ページ) を参照してください。

- 汎用リソースを使用してクリティカルリソースをパッケージの一部として監視するには、次のパラメーター([172 ページ](#)) について値を入力します。
 - `generic_resource_name` は、パッケージ内で汎用リソースを識別するために使用します。
 - `generic_resource_evaluation_type` は、汎用リソースのステータスの評価をパッケージの起動時と起動前のどちらで行うのかを定義するために使用します。
 - `generic_resource_up_criteria` は、汎用リソースのステータスを指定した基準に基づき判定するために使用します。

詳細は、「[汎用リソースの構成](#)」(97 ページ) を参照してください。

- パッケージが LVM ボリュームグループをアクティブ化する必要がある場合は、`vgchange_cmd` を設定するか、デフォルトのままにします。
- パッケージがファイルシステム (Red Hat GFS 以外。 [fs_type](#) ([176 ページ](#)) を参照) に LVM ボリュームをマウントする必要がある場合には、`vg` パラメーターを使ってアクティブ化するボリュームグループの名前を指定し、適切な `vgchange_cmd` を選択します。
ファイルシステムの特長、マウント方法、マウント場所を指定するには、`fs_` パラメーター([176 ページ](#)) を使います。詳細と例については、構成ファイルの `FILESYSTEMS` セクションにあるコメントを参照してください。

各ボリュームグループは、次のように、1 行に 1 つずつ指定します。

```
vg vg01
```

```
vg vg02
```

- パッケージで多数のファイルシステムをマウントする場合は、以下のパラメーターの値を大きくすることを検討してください。
 - `concurrent_fsck_operations` - パッケージの起動時に同時に実行を許可する `fsck` の数を指定します (Red Hat GFS では使われません)。
Red Hat GFS は Serviceguard A.11.20.00 ではサポートされません。
- ファイルシステムのマウントとマウント解除の再試行オプションを指定します。Red Hat GFS ([fs_type](#) ([176 ページ](#)) を参照) では、デフォルト (ゼロ) を指定します。
- `pev_` パラメーターを使って、外部スクリプトに渡される変数を指定できます。変数名は大文字または小文字の `pev` とアンダースコア (`_`) で始まることに注意してください。複数の変数を指定できます。詳細は、「[外部スクリプトについて](#)」([115 ページ](#)) および構成ファイル内のコメントを参照してください。
- パッケージの起動/停止時に外部「プレスクリプト」を実行する場合は、`external_pre_script` パラメーター([178 ページ](#)) を使って、スクリプトの絶対パス名を指定します。たとえば、`$SGCONF/pkg1/pre_script1` のように指定します。
- パッケージが外部スクリプトを実行する場合は、`external_script` パラメーター([178 ページ](#)) を使って、スクリプトの絶対パス名を指定します。たとえば、`$SGCONF/pkg1/script1` のように指定します。

詳細は、「[外部スクリプトについて](#)」([115 ページ](#)) および構成ファイル内のコメントを参照してください。

- 最大で 8 人の個別ユーザー、または `any_user` にアクセス制御ポリシーを構成します。パッケージ構成ファイルに構成できるユーザーロールは、そのパッケージに対する `package_admin` だけです。クラスター構成ファイルではクラスター範囲のロールが定義されます。詳細は、「[アクセス制御ポリシーの設定](#)」(148 ページ)を参照してください。

パッケージ構成の検証と適用

Serviceguard は、ユーザーが入力した構成をチェックし、エラーがあればレポートします。たとえば、次のようなコマンドを使って、作成したパッケージ構成ファイルの内容を確認します。

```
cmcheckconf -v -P $SGCONF/pkg1/pkg1.conf
```

エラーは、標準出力に表示されます。必要に応じてファイルを編集してエラーを訂正し、エラーがなくなるまで `cmcheckconf` の実行を繰り返します。

以下の項目が確認されます。

- パッケージ名が有効であること、そして `node_name` のエントリーが少なくとも 1 つあること。
- 重複したパラメーターエントリーがないこと (複数のボリュームグループが許されているような場合は除く)。
- すべてのパラメーターに許容範囲内の値が設定されていること。
- 構成されたリソースはクラスターノード上で利用可能である。
- ファイルシステムとボリュームグループが有効なこと。
- サービスが実行可能なこと。
- このパッケージが依存するパッケージが、クラスター構成にすでに含まれていること。

詳細は、`cmcheckconf (1m)` のマンページおよび「[クラスター構成要素のチェック](#)」(195 ページ)を参照してください。

`cmcheckconf` がエラーがなしで完了した場合には、次のようにして、パッケージ構成を適用します。

```
cmapplyconf -P $SGCONF/pkg1/pkg1.conf
```

これによりパッケージ構成情報が `$SGCONF` ディレクトリ内の バイナリ形式クラスター構成ファイルに追加され、すべてのクラスターノードに配布されます。

注記: モジュラーパッケージでは、`external_pre_script` パラメーターと `external_script` パラメーターで指定した外部スクリプトの配布が必要になりました。

しかし、従来のパッケージの構成方法に慣れている場合には、モジュラーパッケージでは独立したパッケージ制御スクリプトが**不要**なことや、構成ファイルを手作業で配布する必要がなくなったことに注意してください。(従来のパッケージでは、これまでどおりこれらの作業が必要です。「[従来のパッケージの構成](#)」(218 ページ)を参照してください。)

クラスターへのパッケージの追加

クラスターが稼働中でも新しいパッケージをクラスターに追加することができます。ただし、クラスター構成ファイル内の `max_configured_packages` の値を超えて追加することはできません。「[稼働中のクラスターへのパッケージの追加](#)」(227 ページ)を参照してください。

ディスク監視構成の作成

Serviceguard では、クラスター内のパッケージによってアクティブ化された共有ストレージのディスク監視を行うことができます。各ノード上で監視対象として構成したすべてのディスクの状態をそのノード上の監視デーモンが追跡します。

構成はクラスター内の各ノードで個別に行う必要があります。これは、各ノードはそのノードでアクティブ化可能なディスクグループだけを監視し、またアクティブ化可能なディスクグループは、そのノードでどのパッケージの実行が許可されているかに依存するためです。

監視を設定するには、追跡するディスクを使う各パッケージに監視サービスを含めます。サービス名はクラスター内で一意でなければならないことに注意してください。パッケージ名と文字列 `cmresserviced` を組み合わせて使うことができます。以下に、`pkg1` のパッケージ構成ファイル内のエントリーを示します。

```
service_name          cmresserviced_pkg1
service_fail_fast_enabled  yes
service_halt_timeout   300
service_cmd            "cmresserviced /dev/sdd1 /dsv/sde1"
service_restart        none
```



注意: LVM の制限により、`service_fail_fast_enabled` に `yes` を設定し、ストレージにアクセスできなくなったらパッケージを別のノードに強制的にフェイルオーバーさせる必要があります。

注記: `service_cmd` エントリーには `cmresserviced` コマンドを含める必要があります。
`service_restart` に `none` を設定することも重要です。

7 クラスターとパッケージの管理

本章では、`cmviewcl` コマンド、クラスターや各ノードを起動したり停止する方法、恒久的な再構成の方法、クラスターの定常保守でパッケージを起動、停止、移動、変更する方法について、以下の内容を説明します。

- クラスターとパッケージのステータスの確認
- クラスターとノードの管理 (196 ページ)
- パッケージとサービスの管理 (203 ページ)
- クラスターの再構成 (209 ページ)
- 従来のパッケージの構成 (218 ページ)
- パッケージの再構成 (225 ページ)
- クラスターイベント発生時の対処 (232 ページ)
- 単一ノードによる運用 (232 ページ)
- システム上の Serviceguard の削除 (233 ページ)

クラスターとパッケージのステータスの確認

Serviceguard Manager またはクラスターノードのコマンド行から、ステータスをチェックすることができます。

cmviewcl コマンドによるクラスターとパッケージ状態の確認

クラスターのステータスに関する情報は、ステータスデータベースに保存されます。このステータスデータベースは、クラスター内の個々のノードでそれぞれ保持されています。`cmviewcl` コマンドを使って、このデータベースに保存されている情報を表示できます。

`cmviewcl -v`

`cmviewcl` コマンドは、`root` の権限がなくても使用できます。Serviceguard バージョン A.11.16 以降が動作しているクラスターでは、ユーザーにモニターロールを割り当てることでアクセスを許可することができます。以前のバージョンでは、`<nodename> <nonrootuser>` を `cmclnodelist` ファイルに追加することでアクセスを許可していました。

`cmviewcl -v` では、フェイルオーバーの動作を決定するパラメーターの設定とともに、動作しているクラスター内のすべてのノードとパッケージについての情報が表示されます。



ヒント: 大規模な構成の場合、コマンドによっては長時間かかることがあります。特に、パッケージやサービスの数が増えると、`cmviewcl -v` 実行時の Serviceguard の CPU 使用率が増加します。

`cmviewcl` のその他のオプションの詳細については、マンページを参照してください。

パッケージの依存関係の表示

`cmviewcl -v` コマンドを使用すると、クラスター全体の依存関係が表示されます。特定のパッケージの依存関係を表示するには、`-p<pkgname>` オプションを使います。

クラスターのステータス

`cmviewcl` コマンドで示される、クラスターの**ステータス**は、以下のいずれかです。

- `up`。少なくとも 1 つのノードでクラスターデーモンが実行され、再構成は行われていません。
- `down`。クラスターデーモンは、どのクラスターノードでも実行されていません。

- **starting**。クラスターは、アクティブなメンバーの判定処理を行っています。少なくとも 1 つのクラスターデーモンが実行されています。
- **unknown**。cmviewcl コマンドが実行されるノードはクラスター内の他のノードと通信できません。

ノードのステータスと状態

ノードの**ステータス**は、そのノード上でクラスターデーモンが実行されているかどうかによって、Up (クラスターのメンバーとして**アクティブ**である) または Down (クラスター内では**非アクティブ**である) のどちらかになります。ただしクラスターという観点から見ればダウンしている状態のノードでも、Linux は実行されています。

ノードの**状態**は、次のいずれかです。

- **Failed** (実行できない)。**"Unknown"**状態のノードは、ノード自身が**"Unknown"**状態かどうかを認識することはできません。クラスター内の他のアクティブなメンバーであるノードでは、クラスター内のノードが現在はアクティブではなく、かつシャットダウンもされていない場合、そのノードが**"Failed"**の状態であると判断できます。
- **Reforming** (再編成)。クラスターが再編成を行っている時に、ノードはこの状態になります。この状態のノードでは、全ノードが、アクティブなクラスターの新しいメンバー構成を承認するためのプロトコルを実行しています。すべてのノードで承認されると、新しいクラスターメンバーを反映して、ステータスデータベースがアップデートされます。
- **Running** (稼働中)。この状態のノードは、前回の再編成で必要な作業をすべて完了して、正常に動作しています。
- **Halted** (停止状態)。**"Unknown"**状態のノードは、ノード自身が**"Unknown"**状態かどうかを認識することはできません。あるノードが cmhaltnode コマンドなどにより正常に停止され、クラスターから離脱すると、クラスター内の他のアクティブなメンバーであるノードは、そのノードが**"Halted"**状態であると判断します。
- **Unknown** (未定義)。**"Unknown"**状態のノードは、ノード自身が**"Unknown"**状態かどうかを認識することはできません。他のノードはアクティブなクラスターメンバーになったことがないノードを**"Unknown"**状態であると判断します。

パッケージのステータスと状態

パッケージの**ステータス**は、次のいずれかです。

- **up** - パッケージマスター制御スクリプトがアクティブです。
- **down** - パッケージマスター制御スクリプトがアクティブではありません。
- **start_wait** - このパッケージに対して cmrunpkg コマンドを実行中です。パッケージは、このパッケージが依存しているパッケージ (先行パッケージ) が起動されるのを待っています。
- **starting** - パッケージが起動中です。パッケージマスター制御スクリプトが実行されています。
- **halting** - このパッケージに対して cmhaltpkg コマンドが実行されており、停止スクリプトが実行されています。
- **halt_wait** - このパッケージに対して cmhaltpkg コマンドを実行中です。パッケージは停止するのを待っていますが、このパッケージに依存しているパッケージ (後続パッケージ) が停止するのを待っているため停止スクリプトを起動できません。詳細は、[successor_halt_timeout \(166 ページ\)](#) のパラメーターの説明を参照してください。
- **failing** - パッケージは、そのパッケージ自身またはそのパッケージが依存しているパッケージが失敗したため、停止しています。

- `fail_wait` - パッケージは、そのパッケージ自身またはそのパッケージが依存しているパッケージが失敗したため停止する必要がありますが、その前にそのパッケージに依存しているパッケージが停止するのを待っています。
- `relocate_wait` - パッケージの停止スクリプトが完了したか、Serviceguard がまだパッケージの配置を試みています。
- `reconfiguring` - このパッケージが動作しているノードは、適用された最新の変更を反映するためにパッケージ構成を調整中です。
- `reconfigure_wait` - このパッケージが動作しているノードは、適用された最新の変更を反映するためにパッケージ構成の調整を待機中です。
- `detached` - クラスターまたはノードが `d` オプションで停止されるときに、パッケージが、実行されていたそのクラスターまたはノードから切り離されたことを示しています。Serviceguard はこのパッケージをこれ以上監視しません。パッケージがクラスターから切り離される前、最後に認識されたパッケージのステータスは `Up` でした。
- `unknown` - `cmviewcl` が実行された時点で、Serviceguard がステータスを判別できませんでした。

システムマルチノードパッケージは、アクティブな**すべての**クラスターノードで動作している場合にアクティブになります。マルチノードパッケージは、構成済みの**いずれかの**ノードで動作している場合にアクティブになります。

システムマルチノードパッケージは、`changing` のステータスが可能です。これは、1 つまたは複数のアクティブノードでパッケージが移行中であることを示します。

パッケージの**状態**は、次のいずれかです。

- `starting` - パッケージが起動中です。パッケージマスター制御スクリプトが実行されています。
- `start_wait` - このパッケージに対して `cmrunpkg` コマンドを実行中です。パッケージは、このパッケージが依存しているパッケージ(先行パッケージ)が起動されるのを待っています。
- `running` - サービスがアクティブになっていて、監視されています。
- `halting` - このパッケージに対して `cmhaltpkg` コマンドが実行されており、停止スクリプトが実行されています。
- `halt_wait` - このパッケージに対して `cmhaltpkg` コマンドを実行中です。パッケージは停止するのを待っていますが、このパッケージに依存しているパッケージ(後続パッケージ)が停止するのを待っているため停止スクリプトを起動できません。詳細は、[successor_halt_timeout \(166 ページ\)](#) のパラメーターの説明を参照してください。
- `halted` - パッケージが終了し、停止しました。
- `failing` - パッケージは、そのパッケージ自身またはそのパッケージが依存しているパッケージが失敗したため、停止しています。
- `fail_wait` - パッケージは、そのパッケージ自身またはそのパッケージが依存しているパッケージが失敗したため停止する必要がありますが、その前に依存しているパッケージが停止するのを待っています。
- `failed` - パッケージは失敗し、終了しました。
- `relocate_wait` - パッケージの停止スクリプトが完了したか、Serviceguard がまだパッケージの配置を試みています。
- `maintenance` - パッケージは保守モードにあります。[「パッケージの保守: 保守モード」 \(205 ページ\)](#) を参照してください。
- `detached` - パッケージが動作しているクラスターまたはノードが `d` オプションで停止されるときに、パッケージがクラスターから切り離されたことを示しています。パッケージ

が切り離されると、すべてのパッケージコンポーネントは Up であり稼働しています。Serviceguard は、detached 状態のパッケージを監視しません。

- reconfiguring - このパッケージが動作しているノードは、適用された最新の変更を反映するためにパッケージ構成を調整中です。
- reconfigure_wait - このパッケージが動作しているノードは、適用された最新の変更を反映するためにパッケージ構成の調整を待機中です。
- unknown - cmviewcl が実行された時点で、Serviceguard が状態を判別できませんでした。

以下の状態は、マルチノードパッケージでのみ起こります。

- blocked - パッケージの依存関係が満たされていない、または auto_run が no に設定されているためパッケージは実行されていません。
- changing - パッケージは、表示されているステータスとは異なり、あるノード上で一時的な状態にあります。たとえば、ステータスが starting で、状態が changing の場合、パッケージは少なくとも1つのノードで起動したが、少なくとも他の1つのノードでは、遷移状況 (たとえば failing) にあることを意味します。

パッケージの切り替え属性

cmviewcl コマンドを使うと、以下のパッケージの切り替え情報が表示されます。

- AUTO_RUN (自動起動): enabled または disabled の値を取ります。フェイルオーバーパッケージの場合は、enabled (有効) とは、クラスターの起動時にパッケージが起動され、障害時に Serviceguard がパッケージを別のノードに切り替えることを意味します。システムマルチノードパッケージの場合は、enabled (有効) とは、クラスターに新しく参加したノードでパッケージのインスタンスが起動できることを意味します (disabled (無効) の場合は起動されません)。
- ノードで有効になっている切り替え: フェイルオーバーパッケージの場合は、enabled (有効) とは、指定されたノードへパッケージが切り替え可能であることを意味します。disabled (無効) とは、cmmodpkg コマンドを使用してノードでパッケージが実行できるようにしない限り、パッケージは指定されたノードに切り替えられないことを意味します。
どのフェイルオーバーパッケージについても、パッケージのプライマリノードまたは引き継ぎノードのそれぞれに対して enabled (有効) または disabled (無効) のどちらかを設定します。
マルチノードパッケージの場合は、ノード切り替えが disabled (無効) であるとは、パッケージがそのノードでは起動できないことを意味します。

サービスのステータス

サービスのステータスは、次のいずれかです。

- Up。サービスは監視されています。
- Down。サービスは実行されていません。起動されていないか、停止または障害の可能性があります。
- Unknown。Serviceguard がステータスを認識できません。

汎用リソースのステータス

汎用リソースのステータスは、次のいずれかです。

- Up。汎用リソースの状態はアップです。
- Down。汎用リソースの状態はダウンです。

- Unknown。リソース監視がまだリソースのステータスを設定していません。

ネットワークのステータス

ネットワークインターフェイスのステータスは、次のいずれかです。

- Up。
- Down。
- Unknown。Serviceguard がインターフェイスのステータスを認識できません。

フェイルオーバーポリシーとフェイルバックポリシー

フェイルオーバーパッケージの *failover_policy* パラメーター(166 ページ) には、次の 2 つの値のどちらかを指定することができます。cmviewcl -v の出力で確認することができます。

- configured_node: パッケージは、パッケージ構成ファイルの *node_name* リスト(164 ページ) に記述されている次のノードへフェイルオーバーします。
- min_package_node: パッケージは、ノードで実行中のパッケージの数がクラスター内で最も少ないノードへフェイルオーバーします。

また、フェイルオーバーパッケージの *failback_policy* パラメーター(167 ページ) には、次の 2 つの値のどちらかを指定することができます。cmviewcl -v の出力で確認することができます。

- automatic: フェイルオーバー後に、プライマリノードが再び使用可能になった時点で、パッケージはプライマリノードに戻ります。
- manual: フェイルオーバー後に、パッケージは、システム管理者が手動で元のノードに戻すまで引き継ぎノードで実行されます。

クラスターとパッケージの状態の例

次の cmviewcl -v コマンドの出力例は、サンプル構成のクラスターのステータスを示しています。

正常稼働中のステータス

すべてが正常に稼働しています。クラスターの両方のノードが稼働しており、パッケージはプライマリノードで稼働しています。

```
CLUSTER      STATUS
example      up
  NODE      STATUS      STATE
  ftsys9    up          running

  Network_Parameters:
  INTERFACE  STATUS      NAME
  PRIMARY    up          eth0
  PRIMARY    up          eth1

  PACKAGE    STATUS      STATE      AUTO_RUN  NODE
  pkg1       up          running    enabled   ftsys9

  Policy_Parameters:
  POLICY_NAME  CONFIGURED_VALUE
  Failover     configured_node
  Failback     manual

  Script_Parameters:
  ITEM      STATUS      MAX_RESTARTS  RESTARTS  NAME
  Service   up          0             0          service1
Service    up          0             0          sfm_disk_monitor
  Subnet    up          0             0          15.13.168.0
Generic Resource up          0             0          sfm_disk

  Node_Switching_Parameters:
  NODE_TYPE  STATUS      SWITCHING  NAME
```

```

Primary      up      enabled      ftsys9      (current)
Alternate    up      enabled      ftsys10
NODE         STATUS    STATE
ftsys10      up      running

Network_Parameters:
INTERFACE    STATUS    NAME
PRIMARY      up      eth0
PRIMARY      up      eth1

PACKAGE      STATUS    STATE      AUTO_RUN    NODE
pkg2         up      running    enabled     ftsys10

Policy_Parameters:
POLICY_NAME    CONFIGURED_VALUE
Failover       configured_node
Failback       manual

Script_Parameters:
ITEM           STATUS    MAX_RESTARTS  RESTARTS    NAME
Service        up      0             0           0           service2
Service        up      0             0           sfm_disk_monitor 1
Subnet         up      0             0           15.13.168.0
Generic Resource up      0             0           sfm_disk1

Node_Switching_Parameters:
NODE_TYPE      STATUS    SWITCHING    NAME
Primary        up      enabled     ftsys10      (current)
Alternate      up      enabled     ftsys9

```

注記: cmviewcl の PACKAGE 出力の Script_Parameters セクションには、パッケージが実行中のノードにのみ、Subnet のステータスが表示されます。パッケージが他のサブネット上のノードにフェイルオーバーできるクロスサブネット構成では、他のサブネットは表示されません (「[クロスサブネット構成](#)」(25 ページ) を参照)。

クォーラムサーバーのステータス

クラスターがタイブレークサービスにクォーラムサーバーを使用している場合、次の cmviewcl -v の出力のように、各ノードのエントリの下にサーバー名とその状態およびステータスが表示されます。

```

CLUSTER      STATUS
example      up

NODE         STATUS    STATE
ftsys9       up      running

Quorum Server Status:
NAME          STATUS    STATE
lp-qs         up      running
...

NODE         STATUS    STATE
ftsys10      up      running

Quorum Server Status:
NAME          STATUS    STATE
lp-qs         up      running

```

パッケージ停止後のステータス

cmhaltpkg コマンドで pkg2 を停止させた後、cmviewcl -v の出力は以下のようになります。

```

CLUSTER      STATUS
example      up

NODE         STATUS    STATE
ftsys9       up      running

```

```

Network_Parameters:
INTERFACE      STATUS      NAME
PRIMARY        up          eth0
PRIMARY        up          eth1

PACKAGE        STATUS      STATE      AUTO_RUN      NODE
pkg1           up          running    enabled        ftsys9

Policy_Parameters:
POLICY_NAME      CONFIGURED_VALUE
Failover         configured_node
Failback         manual

Script_Parameters:
ITEM      STATUS      MAX_RESTARTS      RESTARTS      NAME
Service   up          0                  0              service1
Service   up          0                  0              0    sfm_disk_monitor
Subnet     up          0                  0              15.13.168.0
Generic Resource up          0                  0              sfm_disk

Node_Switching_Parameters:
NODE_TYPE      STATUS      SWITCHING      NAME
Primary        up          enabled        ftsys9        (current)
Alternate      up          enabled        ftsys10

NODE      STATUS      STATE
ftsys10   up          running

Network_Parameters:
INTERFACE      STATUS      NAME
PRIMARY        up          eth0
PRIMARY        up          eth1

UNOWNED_PACKAGES

PACKAGE        STATUS      STATE      AUTO_RUN      NODE
pkg2           down        unowned    disabled        unowned

Policy_Parameters:
POLICY_NAME      CONFIGURED_VALUE
Failover         configured_node
Failback         manual

Script_Parameters:
ITEM      STATUS      NODE_NAME      NAME
Service   down        ftsys9         service2
Generic Resource up          ftsys9         sfm_disk1
Subnet     up          15.13.168.0
Generic Resource up          ftsys10        sfm_disk1

Node_Switching_Parameters:
NODE_TYPE      STATUS      SWITCHING      NAME
Primary        up          enabled        ftsys10
Alternate      up          enabled        ftsys9

```

pkg2 の現在のステータスは down、パッケージ切り替え (AUTO_RUN) が disabled で、unowned と表示されています。ただし、切り替え (SWITCHING) は両方のノードで有効になっていることに注意してください。したがって、パッケージに対してグローバル切り替え属性 (AUTO_RUN) が再び enabled に設定されると、パッケージはプライマリノード上での起動を試みます。

パッケージを別のノードへ移動した後のステータス

次のコマンドを実行したとします。

```
cmrunpkg -n ftsys9 pkg2
```

cmviewcl -v コマンドの出力は、以下ようになります。

```
CLUSTER      STATUS
example      up

      NODE      STATUS      STATE
      ftsys9    up          running

      Network_Parameters:
      INTERFACE  STATUS      NAME
      PRIMARY    up          eth0
      PRIMARY    up          eth1

      PACKAGE    STATUS      STATE      AUTO_RUN      NODE
      pkg1       up          running    enabled        ftsys9

      Policy_Parameters:
      POLICY_NAME CONFIGURED_VALUE
      Failover    configured_node
      Failback    manual

      Script_Parameters:
      ITEM      STATUS  MAX_RESTARTS  RESTARTS      NAME
      Service   up      0             0             service1
Service        up      0             0             0             sfm_disk_monitor
      Subnet     up      0             0             15.13.168.0
Generic Resource up      0             0             sfm_disk

      Node_Switching_Parameters:
      NODE_TYPE  STATUS      SWITCHING      NAME
      Primary    up          enabled        ftsys9          (current)
      Alternate  up          enabled        ftsys10

      PACKAGE    STATUS      STATE      AUTO_RUN      NODE
      pkg2       up          running    disabled      ftsys9

      Policy_Parameters:
      POLICY_NAME CONFIGURED_VALUE
      Failover    configured_node
      Failback    manual

      Script_Parameters:
      ITEM      STATUS  MAX_RESTARTS  RESTARTS      NAME
      Service   up      0             0             0             service2
Service        up      0             0             0             sfm_disk_monitor
      Subnet     up      0             0             15.13.168.0
Generic Resource up      0             0             sfm_disk

      Node_Switching_Parameters:
      NODE_TYPE  STATUS      SWITCHING      NAME
      Primary    up          enabled        ftsys10
      Alternate  up          enabled        ftsys9          (current)

      NODE      STATUS      STATE
      ftsys10    up          running

      Network_Parameters:
      INTERFACE  STATUS      NAME
      PRIMARY    up          eth0
      PRIMARY    up          eth1
```

Package Switching (パッケージ切り替え) が Enabled に設定された後のステータス

次のコマンドを実行すると、パッケージのステータスが Auto Run Enabled に変わります。

cmmodpkg -e pkg2

cmviewcl コマンドの出力は、以下ようになります。

```
CLUSTER      STATUS
example      up
```

NODE	STATUS	STATE		
ftsys9	up	running		

PACKAGE	STATUS	STATE	AUTO_RUN	NODE
pkg1	up	running	enabled	ftsys9
pkg2	up	running	enabled	ftsys9

NODE	STATUS	STATE
ftsys10	up	running

ftsys9 上では現在両方のパッケージが稼働中で、pkg2 は切り替え可能です。ftsys10 はデーモンを実行しており、パッケージは ftsys10 では稼働していません。

ノード停止後のステータス

次のコマンドを実行して ftsys10 を停止させます。

cmhaltnode ftsys10

cmviewcl の出力は、ftsys9 上では以下ようになります。

CLUSTER	STATUS
example	up

NODE	STATUS	STATE
ftsys9	up	running

PACKAGE	STATUS	STATE	AUTO_RUN	NODE
pkg1	up	running	enabled	ftsys9
pkg2	up	running	enabled	ftsys9

NODE	STATUS	STATE
ftsys10	down	halted

ftsys9 と ftsys10 のどちらで実行してもこの出力を得ることができます。

所有されていない (unowned) パッケージの情報の表示

現在の状態が unowned (所有されていない) のパッケージ、つまり構成されているノード上で実行されていないパッケージの例を以下に示します。

UNOWNED_PACKAGES

PACKAGE	STATUS	STATE	AUTO_RUN	NODE
PKG3	down	halted	enabled	unowned

Policy_Parameters:

POLICY_NAME	CONFIGURED_VALUE
Failover	min_package_node
Failback	automatic

Script_Parameters:

ITEM	STATUS	NODE_NAME	NAME
Subnet up	manx	192.8.15.0	
Generic Resource unknown	manx		sfm_disk
Subnet up	burmese	192.8.15.0	
Generic Resource unknown	burmese		sfm_disk
Subnet up	tabby	192.8.15.0	
Generic Resource unknown	tabby		sfm_disk
Subnet up	persian	192.8.15.0	
Generic Resource unknown	persian		sfm_disk

Node_Switching_Parameters:

NODE_TYPE	STATUS	SWITCHING	NAME
Primary	up	enabled	manx
Alternate	up	enabled	burmese

Alternate	up	enabled	tabby
Alternate	up	enabled	persian

クラスター構成要素のチェック

次の表に、ユーザーがチェックできる各構成要素、使用できるコマンドまたはツール、および詳細情報の入手先を示します。

注記: 表では、新しいものだけではなく、A.11.20.00 の段階でチェックできるすべての項目を示します。

表 10 クラスター構成要素の確認

構成要素 (コンテキスト)	ツールまたはコマンド、詳細情報	コメント
ボリュームグループ (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m) 「 パッケージ構成の確認 」(223 ページ)も参照してください。	パッケージ検証の一環としてモジュラーパッケージだけがチェックされます (cmcheckconf -P)。
LVM 論理ボリューム (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m) 「 パッケージ構成の確認 」(223 ページ)も参照してください。	パッケージ検証の一環としてモジュラーパッケージだけがチェックされます (cmcheckconf -P)。
LVM 物理ボリューム (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m)	パッケージ検証の一環としてモジュラーパッケージだけがチェックされます (cmcheckconf -P)。
LAN(クラスター)	cmviewcl (1m)、cmcheckconf (1m)、cmapplyconf (1m) 「 クラスターとパッケージのステータスの確認 」(186 ページ)も参照してください。	それぞれの単一 NIC についてスタンバイのステータスがチェックされます。 cmviewcl 既存のクラスターでは、オフラインスタンバイのフラグが付けられますが、作成中のクラスターには付けられません。
クォーラムサーバー (クラスター)	cmcheckconf (1m)、cmapplyconf (1m) .	クォーラムサーバーが使用されている場合、コマンドは、このクォーラムサーバーが稼働しているかどうかとすべてのノードがこのクォーラムサーバーにアクセスする権限があるかどうかを確認します。また、複数の IP アドレスが指定されている場合は、すべてのノードから両方の IP アドレス経由でクォーラムサーバーにアクセスできるかどうかを確認されます。
ロック LUN (クラスター)	cmcheckconf (1m)、cmapplyconf (1m)	コマンドは、すべてのノードが同じ物理デバイスにアクセスしているかどうかとロック LUN のデバイスファイルがブロック型デバイスファイルであるかどうかを確認します。
		クラスター内のすべてのノードを対象にファイルの整合性をチェックするには、以下の手順に従ってください。

表 10 クラスター構成要素の確認 (続き)

構成要素 (コンテキスト)	ツールまたはコマンド、詳細情報	コメント
マウントポイント (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m) 「パッケージ構成の確認」(223 ページ)も参照してください。	コマンドは、パッケージ構成ファイルで指定されているマウントポイントディレクトリが、パッケージを実行できるすべてのノード上に存在するかどうかをチェックします。
サービスコマンド (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m) 「パッケージ構成の確認」(223 ページ)も参照してください。	コマンドは、サービスコマンドで指定されたファイルが実在し実行可能かどうかをチェックします。あるサービスコマンドのパスがマウントを解除された共有ファイルシステム内部でネストされている場合、そのサービスコマンドはチェックされません。
IP アドレス (クラスター)	cmcheckconf (1m)、cmapplyconf (1m)	コマンドはクラスターに組み込まれているすべての IP アドレスが各ノードの /etc/hosts に記録されているかどうかをチェックします。
パッケージ IP アドレス (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m) 「パッケージ構成の確認」(223 ページ)も参照してください。	
ファイルシステム (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m) 「パッケージ構成の確認」(223 ページ)も参照してください。	LVM の場合に限り、コマンドは、 <i>fs_name</i> パラメーター(176 ページ)で識別されるファイルシステムが論理ボリューム上に構築されているかどうかをチェックします。
汎用リソース (パッケージ)	cmcheckconf (1m)、cmapplyconf (1m) 「パッケージ構成の確認」(223 ページ)も参照してください。	
外部スクリプトとプリスクリプト (モジュラーパッケージ)	cmcheckconf (1m)、cmapplyconf (1m)	スクリプトから「0」以外の値が返されると、それによりコマンドは失敗します。

クラスターとノードの管理

ここでは、以下の作業について説明します。

- 全ノードが停止しているときのクラスターの起動 (197 ページ)
- 稼働中のクラスターへの、構成済みノードの追加 (197 ページ)
- 稼働中のクラスターからのメンバーノードの削除 (198 ページ)
- クラスター全体の停止 (198 ページ)
- クラスターの自動再起動 (198 ページ)
- パッケージを実行した状態でのノードまたはクラスターの停止 (199 ページ)

Serviceguard A.11.16 以降では、適切な権限を持つ非ルートユーザーがこれらの作業を実行することができます。アクセスポリシーの構成についての詳細は、[クラスターへのアクセスの制御](#) (146 ページ) を参照してください。

Serviceguard Manager または Serviceguard コマンド行を使用して、クラスターの起動や停止、ノードの追加や停止を行うことができます。クラスターの起動は、クラスター内の 1 つ以上の

ノード上でクラスターデーモンを実行することを意味します。すべてのノードが現在停止しているか (つまり、クラスターデーモンが動作していない)、個々のノードでクラスターデーモンを起動するかによって、クラスターを起動するために使う Serviceguard コマンドは異なります。

本章では、構成済みのノードをクラスターに追加することと、新しいノードをクラスター構成に追加することを区別しています。構成済みのノードは、クラスター構成ファイルにすでに入力されているノードです。新しいノードは、クラスター構成ファイルを変更することによってクラスターに追加されます。

注記: クラスターや個々のノードを手作業で起動/停止するときには、クォーラムサーバーが構成されている場合でも、クォーラムサーバーへアクセスする必要はありません。クォーラムサーバーは、クラスターの分割が発生しタイブレークが必要になった場合にだけ使われます。

全ノードが停止しているときのクラスターの起動

すべてのクラスターノードが停止している場合、クラスターを起動するには、Serviceguard Manager またはここで説明する `cmrunc1` コマンドを使います。特定の条件でクラスターを起動するために、コマンドオプションが用意されています。

`-v` オプションを指定すると最も多くの情報が出力されます。次のコマンドでは、接続性の確認を行わずに、クラスター内に構成されているすべてのノードが起動されます。

```
cmrunc1 -v
```

`-w` オプションを指定すると、`cmrunc1` は、クラスターの全ノード間の LAN の接続性を確認します。このオプションを省略すると、クラスターがより早く起動しますが、接続性は確認されません。次のコマンドを実行すると、接続性を確認しながら、クラスター内に構成されているすべてのノードが起動されます。

```
cmrunc1 -v -w
```

特定のノードグループを指定するには、`-n` オプションを使います。このオプションを指定しない場合、すべてのノードが起動されます。次の例では、`ftsys9` と `ftsys10` 上でのみ、□-カルに構成されているクラスターを起動します。(この形式のコマンドは、クラスターがどのノードでもまだ実行されていない場合にのみ使うことができます。)

```
cmrunc1 -v -n ftsys9 -n ftsys10
```

△ **注意:** 一部のクラスターノードですでにクラスターが実行されているときに、`cmrunc1 -n` コマンドでクラスターを起動すると、HP Serviceguard はデータの完全性を保証できません。ノード間のネットワーク接続がダウンしている場合、`cmrunc1 -n` を使うと 2 つ目のクラスターが形成されることがあり、他方のクラスター上ですでに実行されているアプリケーションと同じアプリケーションがこの 2 つ目のクラスターで起動されてしまうことがあります。この結果、2 つのアプリケーションが、ディスク上のデータを互いに上書きしてしまう可能性があります。

稼働中のクラスターへの、構成済みノードの追加

稼働中のクラスター内の構成済みノードを起動するには、Serviceguard Manager または次に示す HP Serviceguard のコマンドを使います。

すでに稼働中のクラスターへ 1 つ以上のノードを追加するには、`cmrunnode` コマンドを使います。追加するノードは、クラスターのメンバーとして構成されていなければなりません。次の例では、ノード `ftsys9` と `ftsys10` だけで起動されたクラスターに、ノード `ftsys8` を追加します。`-v`(詳細) オプションではすべてのメッセージが印刷されます。

```
cmrunnode -v ftsys8
```

デフォルトでは、`cmrunnode` はネットワークの確認を行います。それにより、実際のネットワークの設定が、構成されたネットワークの設定に一致することが確認されます。この方法を使うことをお勧めします。最近ネットワークを確認したことがあり、確認に時間がかかることが分かっている場合は、`-w none` オプションを指定して確認をバイパスすることができます。

このノードのクラスターはすでに稼働しているため、ノードはクラスターに結合し、パッケージ構成に応じてパッケージが開始されます ([node_name\(164 ページ\)](#) を参照)。クラスターが稼働中であることをノードが検出できないか、ノードがクラスター構成のメンバーでない場合、このコマンドは失敗します。

稼働中のクラスターからのメンバーノードの削除

ノードをクラスターの運用から外すには、Serviceguard Manager または次に示す Serviceguard コマンドを使います。この操作では、クラスターデーモンを停止することでそのノードがクラスターの運用から外れますが、クラスター構成は変更されません。クラスター構成から恒久的にノードを削除するには、クラスター構成ファイルを作成し直す必要があります。次の項を参照してください。

ノードを停止させると、パッケージを他のノード上で利用できる状態に保ったまま、システムメンテナンスのためにノードを停止させることが容易にできます。メンテナンス終了後、パッケージはプライマリノードに戻すことができます。「[フェイルオーバーパッケージの移動](#)」([204 ページ](#)) を参照してください。

ノードをクラスターに戻すには、cmrunnode を使います。

注記: Linux の shutdown コマンドを実行する前に、ノードをクラスターのメンバーから外すことをお勧めします (以下のように cmhaltnode を実行するか、Serviceguard Manager で [ノードの停止] を選択します)。特に、シャットダウン中にパッケージアプリケーションで障害が発生し、正常に停止しない可能性がある場合には、この操作を行うようにしてください。

Serviceguard コマンドによる、稼働中のクラスターからのメンバーノードの削除

クラスター内の 1 つ以上のノードを停止するには、cmhaltnode コマンドを使います。指定したノード上のクラスターデーモンが停止し、そのノードはそのクラスターのアクティブなノードではなくなります。

パッケージが動作しているノードを停止するには、-f オプションを使います。引き継ぎノードへ切り替え可能なパッケージが実行されている場合、切り替えが実行され、パッケージが引き継ぎノード上で起動されます。たとえば、次のコマンドは、構成例のノード ftsys9 上で実行されている Serviceguard デーモンを停止し、ftsys9 上で実行されているパッケージを ftsys10 に移動します。

```
cmhaltnode -f -v ftsys9
```

これにより各パッケージのマスター制御スクリプト内の停止命令が実行され、ftsys9 ノード上で実行されているパッケージが停止します。ftsys9 が停止し、引き継ぎノード ftsys10 上でパッケージが起動されます。

クラスター全体の停止

稼働中のクラスターを停止するには、Serviceguard Manager または次に示す Serviceguard コマンドを使います。

cmhaltcl コマンドを使って、クラスター全体を停止します。このコマンドを使うと、クラスター構成内のすべてのノードで、HP Serviceguard デーモンが停止します。-f オプションを使うと、パッケージが動作中でもクラスターの停止を強制することができます。このコマンドは、動作中の任意のノードで実行できます。例:

```
cmhaltcl -f -v
```

これにより、すべてのクラスターノードが停止します。

クラスターの自動再起動

長期間の電源異常のように、クラスター内の全ノードが停止するような事故の後に、自動的に再起動するようにクラスターを構成することができます。自動的に再起動する場合は、\$SGAUTOSTART ファイル ([「Serviceguard のファイルの位置」\(122 ページ\)](#) を参照) で AUTOSTART_CMCLD に 1 を設定します。

パッケージを実行した状態でのノードまたはクラスターの停止

ノードまたはクラスター全体を停止して保守を行うときに、影響を受けるパッケージの停止やフェイルオーバーを行いたくない場合があります。このような保守にはノード (1 つまたは複数) のリブートは含まれないこともあります。ハートビートを中断するネットワーク変更が行われる可能性は高いです。

Serviceguard A.11.20.00 の新しいコマンドオプション (まとめて Live Application Detach (LAD) と呼ばれる) を使用すると、パッケージを実行したままでこのような保守を行うことができます。パッケージは Serviceguard によって監視されませんが、アプリケーションは実行し続けます。このような状態のパッケージは切り離された (デタッチされた) パッケージと呼ばれます。

必要な保守が終了すると、ノードまたはクラスターを再起動することができ、パッケージに対する通常の監視が再開します。

注記: なお、LAD 機能は、1 つまたは複数のノードあるいはクラスター全体に対する保守作業を可能にすることを目的としています。保守の対象が個々のパッケージの場合や、クラスターの構成要素の場合でも影響を受けるパッケージが 1 つだけかまたは少数であれば、パッケージ保守モードを使用できます。「[パッケージの保守: 保守モード](#)」(205 ページ) を参照してください。

提供されている機能

- 実行中のパッケージを停止またはフェイルオーバーせずに、ノードを停止します (cmhaltnode (1m) に -d オプションを指定)。
ノードを再起動 (cmrunnode (1m)) するまで、それらのパッケージは切り離されたままです。つまり、Serviceguard によって監視されません。
- 実行中のパッケージを停止せずに、クラスターを停止します (cmhaltcl (1m) に -d オプションを指定)。
クラスターを再起動 (cmruncl (1m)) するまで、それらのパッケージは切り離されたままです。つまり、Serviceguard によって監視されません。
- 切り離されたパッケージ (切り離されたマルチノードパッケージのインスタンスも含む) を停止します。
- ノードを再起動 (cmrunnode) するか、クラスターを再起動 (cmruncl) することで、通常のパッケージ監視を再開します。
- cmhaltnode (1m) に -f オプションを指定すると、切り離されたノードを強制的に停止することができます。

規則と制限事項

以下の規則と制限があります。

- クラスター内のすべてのノードで Serviceguard A.11.20.00 以降を実行している必要があります。
- 構成されているクラスターノードはすべて、使用可能なネットワークから到達可能である必要があります。
- Live Application Detach でのノードやクラスターの停止または起動、および切り離されたパッケージの停止は、ルートユーザー (スーパーユーザー) で行う必要があります。
- Live Application Detach は、モジュラーフェイルオーバーパッケージとモジュラーマルチノードパッケージだけでサポートされます。
 - クラスター内でシステムマルチノードパッケージが構成されている場合、Live Application Detach は使用できません。

パッケージの種類についての詳細は、[第6章 \(157 ページ\)](#) を参照してください。

- 保守モードにあるパッケージを切り離すことはできません。また、依存関係のあるパッケージのいずれかが切り離されている場合、パッケージを保守モードに移行することはできません。切り離されたパッケージを保守モードに移行することもできません。
保守モードの詳細は、「[パッケージの保守: 保守モード](#)」(205 ページ) を参照してください。依存関係の詳細は、「[パッケージの依存関係について](#)」(100 ページ) を参照してください。
- 切り離されているパッケージを持つパッケージまたはクラスターは、構成変更ができません。
`cmapplyconf (1m)` は失敗します。
- クラスターの状態がダウンのときに、切り離されたパッケージを停止することはできません。
ノードを停止してそのパッケージを切り離している場合、クラスター内で稼働している他のノードにスーパーユーザーでログインし、切り離されたパッケージを停止できます。しかし、クラスターを停止済みの場合、パッケージを停止するには、クラスターを再起動して、パッケージを接続しなおす必要があります。
- `cmeval (1m)` は、Live Application Detach をサポートしません。
`cmeval` についての詳細は、「[クラスターの変更による影響のプレビュー](#)」(210 ページ) を参照してください。
- プレビューモード (-t) では、`cmrunnode` および `cmrunc1` が提供できるのは、再接続されたパッケージの効果の部分的な評価だけです。
評価は、再接続が予定されているパッケージに依存するパッケージの配置を正確には予測しないことがあります。プレビューモードの詳細は、「[クラスターの変更による影響のプレビュー](#)」(210 ページ) を参照してください。
- `cmmodpkg -e -t` は、切り離されたパッケージではサポートされません。
- 切り離されたパッケージを実行できません。
この制限事項は、パッケージが切断されている (そのため、Serviceguard で監視されていない) 間にパッケージの障害を検出すると明らかになります。パッケージを別のノード上で再起動する前に、パッケージの切り離しが行われたノード上で `cmhaltpkg (1m)` を実行してパッケージを停止する必要があります。
- STARTING、HALTING など、遷移状態にあるパッケージを停止することはできません。
パッケージの状態の詳細は、「[パッケージのステータスと状態](#)」(187 ページ) を参照してください。

その他の注意点

次の点に注意してください。

- パッケージを切り離しても、パッケージはそのまま稼働しますが、高可用性のための保護はありません。
Serviceguard は、切り離されたパッケージの構成要素の障害を検出しません。また、パッケージがフェイルオーバーされることもありません。
-
- ① **重要:** つまり、パッケージが切り離されている間は、発生したエラーをユーザーが検出して修正措置を行う必要があります。修正措置を行うには、`cmhaltpkg` を実行して切り離されたパッケージを停止し、`cmrunpkg (1m)` を実行して別のノードでパッケージを再起動してください。
-

- 切り離されたパッケージがあるノードまたはクラスターを再起動すると、パッケージは再接続されます。つまり、Serviceguard がこれらのパッケージの監視を再開します。

Serviceguard は、この時点で切り離されていたパッケージの状態をチェックし、必要な修正措置を実行します。たとえば、切り離されている間にフェイルオーバーパッケージで実際に障害が発生した場合、Serviceguard はそのパッケージを停止して使用できる別のノードで再起動します。

△ 注意: Serviceguard は、パッケージを再接続する際に、LVM ボリュームグループ、マウントポイント、および再配置可能 IP アドレスをチェック**しません**。

- `cmviewc1 (1m)` は、切り離されたパッケージのステータスおよび状態を `detached` として報告します。
パッケージが切り離された後で問題が発生し、パッケージ構成要素の一部あるいはすべてが正常に稼働しているわけではない場合であっても、このように報告されます。
- Serviceguard は切り離されたパッケージが正常な状態を維持していると見なすため、パッケージは依存関係において UP と見なされます。
これは、たとえば、`pkgA` を切り離して ノード 1 を停止し、また、`pkgB` が、`ANY_NODE` で UP の `pkgA` に依存している場合、`pkgA` が切り離されている間も、ノード 2 上の `pkgB` は稼働し続ける(または起動できる)ことを意味します。依存関係の詳細は、「[パッケージの依存関係について](#)」(100 ページ)を参照してください。
- パッケージは、通常の状態と同じく、停止されたノードやクラスターでは起動できません。
- 切り離されたパッケージを持つノードがリブートし、その後復帰すると、以下のような可能性があります。
 - クラスターに再度参加します。切り離されたパッケージの状態は `"running"` または `"failed"` に変わります。切り離されたパッケージの状態が `"running"` に変わった場合は、リブート後に何らかの不整合が発生している可能性があるため、停止して再実行する必要があります。
 - クラスターに再度参加しません。切り離されたパッケージは切り離されたままになります。このようなパッケージは、リブートが原因で発生する不整合を避けるため、停止して再実行する必要があります。
- あるパッケージを停止して無効にした後で、`cmhaltc1 -d` を実行してクラスター内で実行されている別のパッケージを切り離すと、クラスターの再起動時に、このパッケージの `auto_run` が自動的に有効に切り替えられ、パッケージが強制的に起動されます。
この動作を阻止して、クラスターの再起動後に、パッケージを停止および無効のままにするには、`cmhaltc1 -d` を実行する前に、パッケージ構成ファイル([164 ページ](#))で `auto_run` を `no` に変更してパッケージに適用しなおします。

ノードの停止とそのパッケージの切り離し

ノードを停止してそのパッケージを切り離すには、以下の手順に従ってください。

1. 「[規則と制限事項](#)」(199 ページ)で説明する条件が満たされていることを確認します。
2. 従来のパッケージ、システムマルチノードパッケージなど、Live Application Detach の条件を満たしていないパッケージを停止します。

例:

```
cmhaltpkg -n node1 legpak1 legpak2
```

注記: この操作を行わないと、次の手順にある `cmhaltnode` が失敗します。

3. `-d (detach)` オプションを使用してノードを停止します。

```
cmhaltnode -d node1
```

注記: `-d` オプションと `-f` オプションを同時に使用することはできません。詳細は、`cmhaltnode (1m)` を参照してください。

パッケージを再接続するには、ノードを再起動します。

```
cmrunnode -n node1
```

切り離されたパッケージの停止

`node1` で切り離されたパッケージを停止するには、以下の手順に従ってください。

1. クラスター内で稼働している別のノードにスーパーユーザーでログインします。
2. 次のように入力して、パッケージを停止します。

```
cmhaltpkg -n node1 pkg1
```

クラスターの停止とそのパッケージの切り離し

1. 「規則と制限事項」(199 ページ) で説明する条件が満たされていることを確認します。
2. 従来のパッケージ、システムマルチノードパッケージなど、Live Application Detach の条件を満たしていないパッケージを停止します。

例:

```
cmhaltpkg legpak1 legpak2 legpak3 smnp1
```

注記: この操作を行わないと、次の手順にある `cmhaltcl` が失敗します。

3. `-d (detach)` オプションを使用してクラスターを停止します。

```
cmhaltcl -d
```

注記: `-d` オプションと `-f` オプションを同時に使用することはできません。詳細は、`cmhaltcl (1m)` を参照してください。

パッケージを再接続するには、クラスターを再起動します。

```
cmruncl
```

例: ハートビートサブネットでの保守のためのクラスターの停止

この例では、ネットワークの保守が必要になっているとします。この保守作業により、クラスターのすべてのハートビートサブネットが停止しますが、保守作業の間も、パッケージは稼働し続けます。この例では、パッケージ `pkg1`～`pkg5` は Live Application Detach でサポートされておらず、`pkg6`～`pkgn` はサポートされているとします。

以下の手順を実行します。

1. サポートされていないすべてのパッケージを停止します。

```
cmhaltpkg pkg1 pkg2 pkg3 pkg4 pkg5
```

2. クラスターを停止して、残りのパッケージを切り離します。

```
cmhaltcl -d
```

3. 必要に応じて、ハートビートネットワークをアップグレードします。

4. クラスターを再起動すると、`pkg6`～`pkgn` は自動的に再接続され、他のパッケージも、パッケージ構成ファイルで `auto_run (164 ページ)` が `yes` に設定されていれば起動します。

```
cmruncl
```

5. 次のように入力して、残りのパッケージを起動します。

```
cmmodpkg -e pkg1 pkg2 pkg3 pkg4 pkg5
```

パッケージとサービスの管理

ここでは、以下の作業について説明します。

- [パッケージの起動 \(203 ページ\)](#)
- [パッケージの停止 \(203 ページ\)](#)
- [フェイルオーバーパッケージの移動 \(204 ページ\)](#)
- [パッケージ切り替え動作の変更 \(204 ページ\)](#)

これらの作業は、適切な権限をもった非ルートユーザーが実行できます。アクセスポリシーの構成についての詳細は、[クラスターへのアクセスの制御 \(146 ページ\)](#) を参照してください。

Serviceguard Manager または Serviceguard コマンド行を使って、これらの作業を実行できます。

パッケージの起動

通常、クラスターの一部として構成されているパッケージは、クラスターの起動時に一次ノードで起動されます。パッケージを手操作で停止した場合には、パッケージは手操作で起動する必要があります。この操作は、Serviceguard Manager または以下で説明する Serviceguard コマンドのいずれかを使って行うことができます。

クラスターは稼働している必要があります。また、このパッケージが他のパッケージに依存している場合、他のパッケージはすでに実行されているか、このパッケージを起動するコマンドと同じコマンドで起動されなければなりません (以降の項と、[「パッケージの依存関係について」 \(100 ページ\)](#) を参照してください)。

パッケージを起動するには、Serviceguard Manager を使うか、以下で説明する Serviceguard コマンドを使います。

まず `cmrunpkg` コマンドを使って特定のノードでパッケージを起動し、次に `cmmodpkg` コマンドを使ってパッケージの切り替えを有効にします。以下に例を示します。

```
cmrunpkg -n ftsys9 pkg1
```

```
cmmodpkg -e pkg1
```

この例では、`ftsys9` でパッケージを起動してからパッケージを切り替え可能にしています。パッケージは、停止すると切り替えが無効になるため、パッケージが以前にどこかのノードで停止されている場合は、この手順が必要になります。

依存関係のあるパッケージの起動

パッケージを起動する前に、`cmviewcl` コマンドを使ってパッケージの依存関係をチェックすることをお勧めします。

パッケージは、依存しているすべてのパッケージが実行されていない限り、起動できません。起動しようとする、操作失敗の理由を示す Serviceguard メッセージが出力され、パッケージは起動されません。

このような状況になった場合は、このパッケージが依存しているパッケージを含めて、実行コマンドを再実行します。Serviceguard は、すべてのパッケージを正しい順序で起動します。

パッケージの停止

パッケージは停止させるが、ノードは動作させたままにしたい場合は、パッケージを停止します。

パッケージの停止とノードの停止とは、その効果が異なります。ノードを停止した場合、そのノードのパッケージは引き継ぎノードに切り替わります (切り替えが有効な場合)。パッケー

ジを停止した場合は他のノードへの切り替えは無効になるため、別のノードまたは同じノードで手動により再起動しなければなりません。

システムマルチノードパッケージはすべてのクラスターノードで同時に動作します。これらのパッケージを停止すると、すべてのノードでの実行が停止します。マルチノードパッケージは、複数のノードで同時に動作させることができます。マルチノードパッケージは動作中のすべてのノードまたは指定する一部のノードで実行を停止させることができます。

パッケージを停止するには、Serviceguard Manager を使うか、次の例のように `cmhaltpkg` を使います。

```
cmhaltpkg pkg1
```

これを実行すると `pkg1` が停止され、別のノードへの切り替えが無効になります。

依存関係を持つパッケージの停止

パッケージを停止する前に、`cmviewcl` コマンドを使って、パッケージの依存関係を確認することをお勧めします。

パッケージは、それに依存しているすべてのパッケージが停止しないうちは、停止することはできません。停止を試みても、Serviceguard は操作が失敗した理由を示すメッセージを表示するだけで、パッケージの動作は継続されます。

この場合は、依存している側のパッケージを含めた停止コマンドを再び実行します。Serviceguard は、すべてのパッケージを正しい順番で停止していきます。最初に `cmviewcl` コマンドを使って、**他の**動作中のパッケージが、停止しようとしているパッケージに依存していないことを確認してください。

フェイルオーバーパッケージの移動

あるノードから別のノードへフェイルオーバーパッケージを移動するには、Serviceguard Manager または次に示す Serviceguard コマンドを使います。

フェイルオーバーパッケージを新しいノードに移動する前に、`cmviewcl -v -l package` コマンドを実行して、依存関係を確認することをお勧めします。パッケージが依存関係を持っている場合には、新しいノードで条件が満たされることを確認してください。

パッケージを移動するには、まず、`cmhaltpkg` コマンドを使って移動元ノードでそのパッケージを停止します。こうするとパッケージは、単に停止するだけでなく、パッケージ切り替えも停止されます。

パッケージを停止したら、`cmrunpkg` コマンドを使って新しいノードでパッケージを起動します。そして、以下で説明しているように、切り替えを再度有効にします。

パッケージ切り替え動作の変更

考慮すべきオプションには、次の 2 つがあります。

- パッケージが切り替わる (フェイルオーバーする) かどうか。
- パッケージが特定のノードに切り替わるかどうか。

フェイルオーバーパッケージのパッケージ切り替えが `NO` になっている場合には、パッケージは他のノードに移動できません。また、ノード切り替えが `NO` になっている場合には、パッケージはそのノードには移動できません。マルチノードパッケージでパッケージ切り替えに `NO` が設定されている場合には、パッケージはクラスターに新たに参加するノードでは起動できません。また、ノード切り替えに `NO` が設定されている場合には、パッケージはそのノードでは起動できません。

ノード切り替えとパッケージ切り替えのどちらも、クラスターの稼働中に動的に変更することができます。パッケージ切り替えの初期設定は、パッケージ構成ファイル([164 ページ](#))に設定される `auto_run` パラメーターで決定されます。`auto_run` に `yes` が設定されている場合、パッケージが最初に起動されたときに、パッケージ切り替えが有効になります。ノード切り替

えの初期設定では、パッケージが稼働するように構成されているすべてのノードへの切り替えが許可されています。

パッケージの切り替え動作を変更するには、Serviceguard Manager を使うか、次に示す Serviceguard コマンドを使います。

Serviceguard コマンドを使って、パッケージ切り替え動作を一時的または恒久的に変更できます。

パッケージの実行に関して他のノードへの切り替えを一時的に無効にするには、`cmmodpkg` コマンドを使います。たとえば、`pkg1` が現在実行されている場合、他のノード上での起動を回避するには、次のコマンドを入力します。

```
cmmodpkg -d pkg1
```

このコマンドはパッケージは停止しませんが、そのパッケージの他の場所での起動を無効にします。

`cmmodpkg` コマンドの `-n` オプションを使うと、特定のノードへのパッケージ切り替えを無効にすることができます。次のコマンドは、`pkg1` をノード `lptest3` に切り替えることを無効にします。

```
cmmodpkg -d -n lptest3 pkg1
```

切り替えを恒久的に無効にして、次回クラスターが再起動されたときにもパッケージ切り替えに行った変更が有効なままとするには、パッケージ構成ファイルの `auto_run` フラグを変更し、構成を再度適用します。(「稼働中のクラスターでのパッケージ再構成」(226 ページ)を参照してください。)

パッケージの保守: 保守モード

Serviceguard では、パッケージの稼働中にモジュラーパッケージおよびフェイルオーバーパッケージの構成要素を保守する方法が 2 つ用意されています。(パッケージのタイプとモジュールについての詳細は、[第6章 \(157 ページ\)](#)を参照してください)。これら 2 つの方法は、保守モード、部分起動保守モードと呼ばれます。

注記: ノードまたはクラスター全体の停止を伴う保守作業を実行する必要がある場合は、Live Application Detach を検討してください。「[パッケージを実行した状態でのノードまたはクラスターの停止](#)」(199 ページ)を参照してください。

- 保守モードは、主に、パッケージの稼働中にネットワークを変更する場合に便利です。
「[保守モードを使用した保守の実施](#)」(208 ページ)を参照してください。
- 部分起動保守モードを使用すると、パッケージサービス、ファイルシステム、およびポリシーグループを対象に保守作業を進めることができます。
「[部分起動保守モードを使用した保守の実施](#)」(208 ページ)を参照してください。
- 保守モードも部分起動保守モードも、従来のパッケージや、マルチノードパッケージ、システムマルチノードパッケージでは使用できません。
- パッケージの保守では、パッケージ構成ファイルで指定されている、パッケージの構成は変更されません。
パッケージの再構成の詳細は、「[パッケージの再構成](#)」(225 ページ)を参照してください。

注記: パッケージを部分起動保守モードで実行するには、最初に、パッケージを保守モードに移行する必要があります。つまり、部分起動保守モードのパッケージは、以下で説明する保守モードのパッケージの特性を共有します。また、この 2 つのモードには同じ規則と依存関係規則が適用されます。ただし、部分起動保守モードにはそれ以外の規則が適用され、手順も多くなります。[部分起動保守モードを使用した保守の実施](#)の説明を参照してください。

保守モードまたは部分起動保守モードでのパッケージの実行の特性

Serviceguard は、保守モードのパッケージを他のパッケージとは異なり特別な方法で取り扱いします。保守モードで動作するパッケージには、以下の事項が適用されます。

- Serviceguard は、パッケージサービス、サブネット、汎用リソース、およびファイルシステムが報告した障害を無視します。つまりこれらによってパッケージが異常終了することはありません。

注記: ただし、パッケージ制御スクリプトに障害があると、パッケージの異常終了を**引き起こします**。また、外部スクリプト (またはプリスクリプト) を実行できないとき、またはこのスクリプトが存在しない場合にもパッケージは異常終了します。

- パッケージは自動的にフェイルオーバー、停止、または起動はしません。
- 保守モードのパッケージは、そのパッケージについて構成された (またはデフォルトの) 重みを維持します。これは、その重み (存在する場合) がノードのキャパシティに対してカウントされることを意味します。これは、パッケージが「稼働中」でも「停止中」でも適用されます。(重みとキャパシティについては、「[パッケージの重みについて](#)」(108 ページ) を参照してください。)
- ノード全体およびクラスター全体にわたるイベントは、以下に示すようにパッケージに影響を及ぼします。
 - パッケージが動作するノードが停止またはクラッシュすると、パッケージは保守モードではなくなりますが、自動的に再起動することはありません。
 - クラスターが停止またはクラッシュした場合、クラスターが復帰したときにパッケージは保守モードではなくなります。パッケージ構成ファイルで `auto_run` が `yes` に設定されている場合、Serviceguard はパッケージの起動を試みます。
- `node_fail_fast_enabled`(164 ページ) が `yes` に設定されている場合、次のいずれの状況の下でも Serviceguard はノードを**停止しません**。
 - サブネットの障害
 - 汎用リソースの障害
 - スクリプトが存在しない、またはファイルパーミッションのためにスクリプトを実行できない
 - スクリプトのタイムアウト
 - 再起動カウントの限度を超過している

保守モードまたは部分起動保守モードでのパッケージの規則

- ① **重要:** パッケージの保守のバージョン要件に関する重要情報は、Serviceguard の最新のリリースノートを参照してください。

-
- パッケージを保守モードに移行するには、事前に、パッケージのパッケージ切り替えを無効にしておく必要があります。
 - 複数のノードで、パッケージを保守モードに移行することはできません。
 - ノードはクラスター内でアクティブでなければなりません。また、パッケージを実行する資格を持つ必要があります (パッケージの `node_name` リストに掲載)。
 - パッケージが稼働していない場合、パッケージを保守モードに移行するには、`cmmmodpkg` (1m) の実行時にノード名を指定する必要があります。
 - パッケージが稼働している場合は、パッケージは、そのパッケージが稼働しているノード上でのみ保守モードに移行できます。

- あるノード上でパッケージが保守モードに移行しているときは、そのパッケージを別のノード上で実行することはできません。
- パッケージを保守モードに出し入れすると稼働中の別のパッケージが停止する場合は、パッケージの保守モードへの出し入れはできません。
- 保守モードのときはパッケージの障害は無視されます。このため、稼働中のパッケージは、正常な場合にのみ保守モードから取り出すことができます。
Serviceguard は、パッケージのサービスおよびサブネットの状態をチェックして、パッケージが正常かどうかを判断します。パッケージが正常でない場合、パッケージを保守モードから取り出す前に、パッケージを停止する必要があります。
- パッケージを保守モードから取り出す前に、パッケージ内で構成される汎用リソースを使用できなければなりません (ステータスが「up」)。
- 「[パッケージの再構成](#)」 (225 ページ) のオンライン構成を行うことはできません。
- このパッケージに関連する新しい依存関係を構成することはできません。つまり、このパッケージを別のパッケージに依存させたり、別のパッケージをこのパッケージに依存させたりすることはできません。「[保守モードまたは部分起動保守モードでのパッケージの依存関係規則](#)」 (207 ページ) も参照してください。
- パッケージ操作コマンドの `-t` オプションは、保守モードのパッケージに対しては使用できません。`-t` オプションについての詳細は、「[クラスターの変更による影響のレビュー](#)」 (210 ページ) を参照してください。

部分起動保守モードの追加規則

- 部分起動保守モードからパッケージを取り出すには、事前に、パッケージを停止する必要があります。
- パッケージを部分起動保守モードで実行した後で通常のモードで実行するには、パッケージを保守モードから取り出して、再起動する必要があります。

保守モードまたは部分起動保守モードでのパッケージの依存関係規則

保守モードで動作しているパッケージに関連する新しい依存性を構成することはできません。また以下の規則が適用されます (保守モードのパッケージを `pkgA` と呼びます)。

- `pkgA` に依存するパッケージは、`pkgA` を保守モードにするときに、停止して無効にする必要があります。このことは、「[パッケージの依存関係について](#)」 (100 ページ) に記述されたように、すべてのタイプの依存関係 (排他的依存関係を含む) に適用されます。
 - `pkgA` に依存するパッケージを有効にすることはできません。
 - `pkgA` に依存するパッケージは、依存するパッケージそのものが保守モードにない限り、実行することはできません。
- あるパッケージがUPであることを `pkgA` が要求するような依存関係の規則は無視されるので、これらのパッケージは、`pkgA` が保守モードにある間、必要に応じて停止およびフェイルオーバーが可能です。
- 依存関係にある 2 つのパッケージが保守モードにある場合、これらの 2 つのパッケージに対して依存関係の規則は無視されます。
たとえば、排他的依存関係にある 2 つのパッケージがどちらも保守モードの場合、同時に実行および停止することができます。

注記: 汎用リソースが構成されているパッケージがあり、そのパッケージを保守モードから取り出して稼働状態にしようとする、汎用リソースのステータスが評価されます。汎用リソースのいずれかのステータスが「down」の場合、パッケージを保守モードから取り出すことはできません。

保守モードを使用した保守の実施

パッケージが停止している場合も稼働中の場合も、パッケージを保守モードに移行し、保守を行い、保守モードから取り出すことができます。

このモードは、主に、ネットワークや汎用リソースの構成要素に修正を加える場合に便利です。サービス、ストレージなど、パッケージの他の構成要素を修正する場合は、「[部分起動保守モードを使用した保守の実施](#)」(208 ページ)の追加規則および手順に従ってください。

パッケージを再構成する (cmapplyconf (1m) を使用) 場合は、「[パッケージの再構成](#)」(225 ページ)および「[構成変更が可能なパッケージの状態](#)」(228 ページ)を参照してください。

手順

パッケージのネットワーク構成要素の保守を行う場合は、以下の手順に従ってください。

この例では、パッケージ pkg1 を呼び出します。このパッケージは、node1 上で稼働しているものとします。

1. パッケージを保守モードにします。

```
cmmodpkg -m on -n node1 pkg1
```

2. ネットワークまたはリソースに対して保守を実施し、ネットワークやリソースが正しく機能していることを手動でテストします。

注記: ここで cmviewcl を実行すれば、pkg1 の STATUS が up で、その STATE が maintenance であることがわかります。

3. すべてが期待どおりに機能していれば、パッケージを保守モードから出します。

```
cmmodpkg -m off pkg1
```

部分起動保守モードを使用した保守の実施

パッケージを部分起動保守モードに移行するには、パッケージを保守モードに移行した後で、再起動して、保守作業の対象にしないモジュールだけを起動します。

手順

この手順を使用してパッケージに対する保守を実施します。この例では、パッケージ pkg1 が node1 上で稼働していると仮定します。また、パッケージのサービスに対して保守を実行するとします。

1. パッケージを停止します。

```
cmhaltpkg pkg1
```

2. パッケージを保守モードにします。

```
cmmodpkg -m on -n node1 pkg1
```

注記: 最初の 2 つの手順の順序は逆でもかまいません。

3. 保守モードでパッケージを実行します。

この例では、package_ip までのモジュール (このモジュールも含む) だけが起動されるように pkg1 を起動します。(パッケージモジュールのリストについては、「[パッケージモジュールとパラメーター](#)」(159 ページ)を参照してください。) パッケージが使用するモジュールは、パッケージ構成ファイルの先頭に近い順で起動されます。

```
cmrunpkg -m sg/package_ip pkg1
```

4. サービスに対して保守を実施し、正しくサービスが機能していることを手動でテストします。

注記: ここで cmviewcl を実行すれば、pkg1 の STATUS が up で、その STATE が maintenance であることがわかります。

5. パッケージを停止します。

```
cmhaltpkg pkg1
```

注記: cmhaltpkg -s を使用することもできます。これを使えば、cmrunpkg -m で起動したモジュールが停止します。このケースでは、package_ip までのすべてのモジュールです (このモジュールも含む)。

6. パッケージを実行してすべてが正しく機能していることを確認します。

```
cmrunpkg pkg1
```

注記: パッケージはまだ保守モードのままです。

7. すべてが期待どおりに機能していれば、パッケージを保守モードから出します。

```
cmmodpkg -m off pkg1
```

8. パッケージを再起動します。

```
cmrunpkg pkg1
```

部分起動保守モードでのモジュールの除外

上記の例では、cmrunpkg -m を使用して package_ip までのすべてのモジュール (このモジュールも含む) を実行しますが、それ以降のモジュールはまったく実行しません。しかし、パッケージ内で作業するコンポーネントのモジュールを除いて他のモジュールをすべて実行したい場合があります。この場合には、-e オプションを使用することができます。

```
cmrunpkg -e sg/service pkg1
```

これによってサービスモジュール**以外**のそのパッケージのすべてのモジュールを実行します。

-e は、-m と組み合わせて使用することもできます。これにより、-e で指定されるモジュールを除く、-m で指定されたモジュールまでのすべてのモジュール (このモジュールも含む) を起動することができます。このケースでは、除外 (-e) モジュールの実行順序が -m モジュールよりも早くなければなりません (パッケージ構成ファイルの先頭により近いということです)。

例:

```
cmrunpkg -m sg/services -e sg/package_ip pkg1
```

注記: パッケージを起動するための全モジュールの実行順序は次のとおりです。

1. マスター制御スクリプトそのもの
2. Persistent Reservation

クラスタの再構成

クラスタの再構成は、停止中でも稼働中でも行えます。操作によっては、クラスタの停止中にのみ実行できるものもあります。以下の表に、各種の変更で前提となるクラスタ状態を示します。

表 11 クラスタ構成に対する変更

クラスタ構成の変更	クラスタの状態の必要条件
新しいノードの追加	すべてのクラスタノードが稼働している。
ノードの削除	使用不能または到達不能なノードでも削除できる。
構成済みパッケージの最大数の変更	クラスタは稼働していてもよい。
フォーラムサーバー構成の変更	クラスタは稼働していてもよい。「 オンラインでフォーラム構成を変更したときの動作 」(37 ページ)を参照してください。

表 11 クラスター構成に対する変更 (続き)

クラスター構成の変更	クラスターの状態の必要条件
クラスターロック構成の変更 (ロック LUN)	クラスターは稼働していてもよい。「オンラインでのクラスターロック LUN 構成のアップデート」(217 ページ)と「オンラインでウォーラム構成を変更したときの動作」(37 ページ)を参照してください。
クラスター構成に NIC とその IP アドレスを追加	クラスターは稼働していてもよい。「稼働中のクラスターのクラスターネットワーク構成の変更」(214 ページ)を参照してください。
クラスター構成から NIC とその IP アドレスを削除	クラスターは稼働していてもよい。「稼働中のクラスターのクラスターネットワーク構成の変更」(214 ページ)を参照してください。
既存のインターフェイスの指定を <code>HEARTBEAT_IP</code> から <code>STATIONARY_IP</code> に変更、または その逆	クラスターは稼働していてもよい。「稼働中のクラスターのクラスターネットワーク構成の変更」(214 ページ)を参照してください。
インターフェイスを IPv4 から IPv6 に変更、または その逆	クラスターは稼働していてもよい。「稼働中のクラスターのクラスターネットワーク構成の変更」(214 ページ)を参照してください。
クラスターで使われている NIC の IP アドレスの再構成	インターフェイスをクラスター構成から削除し、再構成し、そして再びクラスター構成に戻す必要がある。「留意事項」(215 ページ)を参照してください。クラスターは稼働したままでもよい。
<code>NETWORK_POLLING_INTERVAL</code> の変更	クラスターは稼働していてもよい。
IP Monitor パラメーター (<code>SUBNET</code> 、 <code>IP_MONITOR</code> 、 <code>POLLING_TARGET</code>) の変更	クラスターは稼働していてもよい。詳細は、「クラスター構成のパラメーター」(82 ページ)で、これらのパラメーターの項目を参照してください。
<code>MEMBER_TIMEOUT</code> および <code>AUTO_START_TIMEOUT</code> の変更	クラスターは稼働していてもよい。
アクセス制御ポリシーの変更	クラスターとパッケージは稼働していてもよい。

クラスターの変更による影響のプレビュー

パッケージの配置には、クラスターノードの可用性、これらのノードでのネットワークやその他のリソースの可用性、フェイルオーバーとフェイルバックのポリシー、およびパッケージの重み/依存関係/優先順位(これらを構成している場合)など、多くの変数が影響します。特定のアクションやイベントがパッケージに及ぼす影響を事前にプレビューすることができます。

たとえば、クラスターを最初に起動したときに、パッケージが想定どおりに配置されるかどうかを確認したい場合があります。また、ノードが停止した場合やノードがその後再起動した場合に、そのノードで稼働しているパッケージがどうなるかをプレビューしたい場合があります。あるいは現在無効なパッケージを有効にした場合や、パッケージが停止して、その `node_list` 上のノードがまったく利用できないために再起動できない場合に、その他のパッケージへの影響を調べたい場合があります。

これに対処するため、Serviceguard には 2 つの方法が用意されています。Serviceguard コマンドのプレビューモードを使用することもできますし、または、`cmeval (1m)` コマンドを使用してクラスターのさまざまな状態をシミュレートすることもできます。

あるいは、クラスターへの変更を全体としてモデル化したい場合があります。`cmeval` を使えば、これが可能となります。「[cmeval の使用](#)」(212 ページ)を参照してください。

プレビュー可能な事象

以下のいずれか、あるいはすべてを同時にプレビューすることができます。

- クラスターの実行 (`cmruncl`)

- クラスターノードの状態変化 (cmrunnode、cmhaltnode)
- パッケージの状態変化 (cmrunpkg、cmhaltpkg)
- あるノードから別のノードへのパッケージの移動
- パッケージの切り替え変化 (cmmodpkg -e)
- パッケージのサブネット、リソース、およびストレージの可用性
- パッケージの優先順位、ノード順序、依存関係、フェイルオーバーとフェイルバックのポリシー、ノードキャパシティ、およびパッケージの重みの変化

Serviceguard Manager でのコマンドのプレビューモードの使用

次のコマンドは `-t` オプションをサポートしているので、プレビューモードでコマンドを実行することができます。

- `cmhaltnode [-t] [-f] <node_name>`
- `cmrunnode [-t] <node_name>`
- `cmhaltpkg [-t] <package_name>`
- `cmrunpkg [-t] [-n node_name] <package_name>`
- `cmmodpkg { -e [-t] | -d } [-n node_name] <package_name>`
- `cmruncl -v [-t]`

注記: 保守モードのパッケージに対して操作するいずれのコマンドについても `-t` オプションを使用することはできません。「[パッケージの保守: 保守モード](#)」(205 ページ) を参照してください。

これらのコマンドの詳細は、それぞれのマンページを参照してください。これらのプレビュー関数を Serviceguard Manager で実行することもできます。Preview [] チェックボックスを各ページの操作で選択します。

`-t` オプションを使用すると、コマンドは、通常の実行とは異なり、**起こりうる**結果を予測し、その概要を `$stdout` に送じます。たとえば、プライマリノードが `node1` である `pkg1` が優先順位の高いパッケージであって、また同じノードで動作する `pkg2` と `pkg3` に依存するものと仮定します。これらは、現在 `node2` で動作する優先順位の低いパッケージです。`pkg1` はダウンして無効になっているので、有効にしたときの影響を見てみましょう。

cmmodpkg -e -t pkg1

次のような出力が得られます。

```
package:pkg3|node:node2|action:failing
package:pkg2|node:node2|action:failing
package:pkg2|node:node1|action:starting
package:pkg3|node:node1|action:starting
package:pkg1|node:node1|action:starting
cmmodpkg: Command preview completed successfully
```

これは、`pkg1` を有効にすると、`pkg1` は、`pkg2` と `pkg3` をプライマリノードである `node1` へ「ドラッグ」していることを示しています。これは、`pkg1` の優先順位が高いため可能です。「[単純依存関係のドラッグ規則](#)」(102 ページ) を参照してください。プレビューを実行することにより、3 つのすべてのパッケージが正しく `node2` で開始されることを確認できます (現在プレビューの時点と、実際に `pkg1` を有効にしたときとで状況に変化がなく、実行スクリプトに障害がないものと仮定)。

注記: プレビューでは、実行スクリプトと停止スクリプトの障害を予測することはできません。

パッケージの依存関係と優先順位についての詳細は、「[パッケージの依存関係について](#)」(100 ページ) を参照してください。

cmeval の使用

cmeval を使えば、Serviceguard パッケージに対するクラスター変化の影響を評価することができます。またこれを使用すれば、実施しようと考えている変更のクラスターへの影響全体についてプレビューのみを行うこともできます。

cmeval は、本番環境で安全に使用することができます。つまりクラスターやパッケージの状態に影響することはありません。コマンドのプレビューモード(上述の `-t`)とは異なり、cmeval は、評価対象のクラスターにログインする必要がなく、実際にクラスターが動作している必要もありませんが、cmeval を実行したシステムと同じリリースで同じパッチバージョンの Serviceguard を使用する必要があります。

単一のコマンドによる影響よりも多くの影響を確認したい場合、特に大規模な変更による結果、あるいは複雑に互いに影響し合う変更(パッケージの優先順位、ノードの順序、依存関係など)による結果を確認したい場合、コマンドのプレビューモードよりも cmeval を使用してください。

cmeval を使用するには、次の 3 つの主要な手順が必要となります。

1. `cmviewcl -v -f line` を使用して現在のクラスター構成をファイルに書き出します。
2. ファイルを編集してプレビューしたいイベントまたは変更を含めます。
3. 手順 2 のファイルを入力として使用し、cmeval を実行して変更の結果をプレビューします。

たとえば、プライマリノードが `node1` である `pkg1` が優先順位の高いパッケージであって、また同じノードで動作する `pkg2` と `pkg3` に依存するものと仮定します。これらの優先順位の低いパッケージは、現在 `node2` で動作しています。`pkg1` はダウンして無効になっているので、有効にしたときの影響を見てみましょう。

`cmviewcl -v -f line` の出力に、行 `package:pkg1|autorun=disabled` があるので、この行を `package:pkg1|autorun=enabled` に変更します。また、パッケージの動作対象として構成されているノードが利用可能であると示す必要があります。たとえば、`package:pkg1|node:node1|available=yes` とします。次にファイルを(たとえば `newstate.in` の名前を)保存し、cmeval を実行します。

cmeval -v newstate.in

次のような出力が得られます。

```
package:pkg3|node:node2|action:failing
package:pkg2|node:node2|action:failing
package:pkg2|node:node1|action:starting
package:pkg3|node:node1|action:starting
package:pkg1|node:node1|action:starting
```

これは、`pkg1` を有効にすると、`pkg1` は、`pkg2` と `pkg3` をプライマリノードである `node1` へ「ドラッグ」していることを示しています。これは、`pkg1` の優先順位が高いため可能です。「[単純依存関係のドラッグ規則](#)」(102 ページ)を参照してください。cmeval を実行することにより、3 つのすべてのパッケージが正しく `node2` で開始されることを確認できます(プレビューの時点と、実際に `pkg1` を有効にしたときとで状況に変化がなく、実行スクリプトに障害がないものと仮定)。

注記: cmeval では、実行スクリプトと停止スクリプトの障害を予測することはできません。

これは簡単な例です。より複雑なシナリオについても、cmeval を使用することができます。「[プレビュー可能な事象](#)」(210 ページ)を参照してください。

- ① **重要:** 詳細および例については、cmeval (1m) のマンページを参照してください。
-

停止したクラスターの再構成

クラスターが停止している間に、そのクラスターの構成を恒久的に変更できます。[表 11](#)の表で「クラスターが稼働していないこと」と記された変更は、必ずこの手順に従って行いますが、その他のクラスター構成も同様の手順で変更できます。

クラスター構成の恒久的な変更を行うには、次の手順に従います。

1. すべてのノードでクラスターを停止します。
2. 1つのノードで、「[HA クラスターの構成](#)」(122 ページ)の説明に従ってクラスターを再構成します。cmgetconf を使ってテンプレートファイルを生成し、それを編集することができます。
3. クラスター構成ファイルに指定されているすべてのノードの電源が入っていて、ノードにアクセスできることを確かめます。cmapplyconf を使ってバイナリ形式クラスター構成ファイルをすべてのノードにコピーします。以前のバージョンのバイナリ形式クラスター構成ファイルは、このファイルで上書きされます。
4. cmruncl を使い、すべてのノードでクラスターを起動するか、一部のノードでクラスターを起動します。

稼働中のクラスターの再構成

クラスターの稼働中に、クラスター構成に新しいノードを追加したり、クラスター構成からノードを削除することができます。ただし、以下の点に注意してください。

- クラスターからアクティブなノードを削除することはできません。先にノードを停止しておかなければなりません。
- 到達できないノード(ネットワークから完全に切り離されているノードなど)に対して唯一可能な構成変更は、そのノードをクラスター構成から削除することです。到達できないノードに依存するパッケージがある場合は、パッケージの構成も変更しないとそのノードを削除することはできません。これらすべては、1 回の構成要求 (cmapplyconf コマンド) で行わなければなりません。

- クラスターのアクセス制御リストは、クラスターの稼働中に変更することができます。

パッケージ構成の変更については、後述する各項を参照してください。

以降の項では、動的な構成作業を行う方法について説明します。

稼働中のクラスターへのノードの追加

ノードを追加するには以下の手順を実行します。この例では、ノード ftsys8 と ftsys9 は稼働中の cluster1 というクラスターにすでに構成されていて、ftsys10 というノードを追加します。

注記: 作業を開始する前に、「[ルートレベルアクセスの構成](#)」(123 ページ)で説明している ftsys10 へのアクセスを構成していることを確認します。

1. 構成の復元が必要になる場合に備え、次のコマンドを使用して、既存のクラスター構成の現在のコピーを一時ファイルに保存します。

```
cmgetconf -C temp.conf
```

2. 構成するノードの新しいセットを指定して、新しい構成のテンプレートを作成します (すべて 1 行で入力)。

```
cmquerycl -C clconfig.conf -c cluster1 -n ftsys8 -n ftsys9 -n ftsys10
```

3. clconfig.conf ファイルをエディターで開いて、新規ノードの情報が正しいか確認します。
4. 新しい構成を検証します。

```
cmcheckconf -C clconfig.conf
```

5. 構成の変更を適用し、新しいバイナリ形式構成ファイルを全クラスターノードに送信します。

```
cmapplyconf -C clconfig.conf
```

cmrunnode コマンドを使って新しいノードを起動し、必要な場合は、\$SGAUTOSTART ファイル(「[Serviceguard のファイルの位置](#)」(122 ページ)を参照)内の AUTOSTART_CMCLD パラメー

ターに 1 を設定して、クラスターをリブートするたびにそのノードが自動的にクラスターに結合するようにします。

稼働中のクラスターからのノードの削除

稼働中のクラスターからノードを削除するには、Serviceguard Manager を使うか、次に示す Serviceguard コマンドを使います。以下の制限があります。

- ノードは停止している必要があります。「稼働中のクラスターからのメンバーノードの削除」(198 ページ) を参照してください。
- 削除するノードが到達不能なノード (LAN から切り離されているノードなど) のときは、そのノードを指定しているパッケージがない場合に限りノードを削除できます。到達不能なノードに依存するパッケージがある場合は、クラスターを停止します。「クラスター全体の停止」(198 ページ) を参照してください。

以下の手順に従い、Serviceguard コマンドを使用してノードを削除します。この例では、ノード `ftsys8`、`ftsys9`、`ftsys10` は稼働中の `cluster1` というクラスターにすでに構成されていて、これからノード `ftsys10` を削除します。

注記: ノードをクラスターから削除するには、同一クラスター内の別のノードから `cmapplyconf` コマンドを実行します。削除するノードでコマンドを実行すると、エラーメッセージが表示されます。

1. 以下のコマンドを使用して、既存のクラスター構成の最新コピーを一時ファイルに保存します。

```
cmgetconf -c cluster1 temp.conf
```

2. 構成するノードの新しいセット (`ftsys10` を除く) を指定して、新しい構成のテンプレートを作成します。

```
cmquerycl -C clconfig.conf -c cluster1 -n ftsys8 -n ftsys9
```

3. `clconfig.conf` ファイルを開いて、クラスターに残っているノードの情報をチェックします。

4. 削除するノード (この例では `ftsys10`) を停止します。

```
cmhaltnode -f -v ftsys10
```

5. 新しい構成を検証します。

```
cmcheckconf -C clconfig.conf
```

6. `ftsys8` または `ftsys9` から構成の変更を適用し、新しいバイナリ形式構成ファイルを全クラスターノードに配布します。

```
cmapplyconf -C clconfig.conf
```

注記: 多くのパッケージが動作するように構成されている到達不能なノードを削除しようとすると、以下のメッセージが表示されることがあります。

The configuration change is too large to process while the cluster is running.

Split the configuration change into multiple requests or halt the cluster.

このような場合は、クラスターを停止してから到達不能なノードを削除する必要があります。

稼働中のクラスターのクラスターネットワーク構成の変更

提供されている機能

実行可能なオンライン操作には、以下のものがあります。

- ネットワークインターフェイスとその `HEARTBEAT_IP` または `STATIONARY_IP` を追加
- ネットワークインターフェイスとその `HEARTBEAT_IP` または `STATIONARY_IP` を削除

- `HEARTBEAT_IP` または `STATIONARY_IP` インターフェイスを IPv4 から IPv6 に変更、または**その逆**
- 既存インターフェイスの指定の `HEARTBEAT_IP` から `STATIONARY_IP` への変更、または**その逆**
- `NETWORK_POLLING_INTERVAL` の変更
- IP Monitor パラメーター (`SUBNET`、`IP_MONITOR`、`POLLING_TARGET`) の変更。詳細は、「[クラスター構成のパラメーター](#)」(82 ページ)にあるこれらのパラメーターの項目を参照してください。
- 1 回の操作 (`cmapplyconf`) での上記の操作の組み合わせ (ただし、以下の制限があります)

留意事項

以下の制限があります。

- すべてのハートビート構成を一度に変更してはなりません。また構成されている唯一のハートビートの変更や削除も行っておりません。
少なくとも 1 つの動作中のハートビートは変更しないでおく必要があります。
- インターフェイスと、クラスター構成内の他のすべてのインターフェイスが正常に動作していない限り、インターフェイスを追加したり、その特性を変更することはできません。
構成内には、不良 NIC や、機能しないサブネットやローカルにスイッチされるサブネットがあってはなりません。ただし、同じ操作でこれらの構成要素を削除するのであれば、あっても構いません。
- クラスター内の他のすべてのノードの、同一サブネット上のすべてのピアネットワークインターフェイスでも同じ変更を行わない限り、既存のインターフェイスの指定を `HEARTBEAT_IP` から `STATIONARY_IP` に変更したり、**その逆**の変更を行うことはできません。
同様に、クラスター内の他のすべてのノードの、同一サブネット上のすべてのピアネットワークインターフェイスでも同じ変更を行わない限り、インターフェイスを IPv4 から IPv6 に変更することはできません。
- すべてのノードでサブネットが共通でない限り、インターフェイスの指定を `STATIONARY_IP` から `HEARTBEAT_IP` に変更することはできません。
`HEARTBEAT_IP` は IPv4 アドレスであり、すべてのノードで同一サブネット上にある必要があります (クロスサブネット構成を除く。「[クロスサブネット構成](#)」(25 ページ)を参照)。
- サブネットや IP アドレスは、それらを (`monitored_subnet`、`ip_subnet`、または `ip_address` として) 使うパッケージがそのノード上で動作するように構成されている場合には、ノードから削除できません。
これらのパラメーターについての情報は、[monitored_subnet](#) (169 ページ) 以降にあります。
- クラスターで使われているインターフェイス (NIC) の IP 構成は、1 回の操作 (`cmapplyconf`) では変更できません。
クラスター構成からまず NIC を削除し、その後 (たとえば、`ifconfig` を使って) NIC を再構成してから、NIC をクラスターに戻す必要があります。
これを行う必要があるのは、次のような場合です。
 - NIC を 1 つのサブネットから別のサブネットに移す
 - NIC に IP アドレスを追加する

- NIC から IP アドレスを削除する

△ **注意:** IP アドレス自体が定常 IP アドレスとしてクラスターにすぐに構成されない限り、Serviceguard クラスターに構成されているネットワークインターフェイスへ IP アドレスを追加しないでください。Serviceguard ネットワークインターフェイスに定常 IP アドレス以外のアドレスを構成すると、Serviceguard が割り当てた再配置可能パッケージアドレスと競合する可能性があります。

いくつかの手順の例を以下に示します。

例: ハートビート LAN の追加

2 ノードクラスター cluster1 のノード ftsys9 と ftsys10 でサブネット 15.13.170.0 が共有されている場合に、そのサブネットをハートビートサブネットとしてクラスター構成に追加する方法について考えます。以下を行ってください。

1. cmquerycl を実行して、クラスター構成に追加可能なインターフェイス用のネットワーク情報を含んでいるクラスター構成テンプレートファイルを取得します。

```
cmquerycl -c cluster1 -C clconfig.conf
```

注記: Serviceguard A.11.18 からは、cmquerycl -c は、クラスター構成に現在は含まれていない利用可能なインターフェイスについて、コメントアウトされたエントリーを含む出力を生成するようになりました。

生成された clconfig.conf のネットワーク関係の編集箇所は、以下のようになります。

```
NODE_NAME ftsys9
NETWORK_INTERFACE lan1
    HEARTBEAT_IP      192.3.17.18
    #NETWORK_INTERFACE lan0
    #STATIONARY_IP    15.13.170.18
NETWORK_INTERFACE lan3
NODE_NAME ftsys10
NETWORK_INTERFACE lan1
    HEARTBEAT_IP      192.3.17.19
    #NETWORK_INTERFACE lan0
# STATIONARY_IP      15.13.170.19
NETWORK_INTERFACE lan3
```

2. ファイルを編集して、追加するサブネット (この例では、lan0) のエントリーのコメントを外し、STATIONARY_IP を HEARTBEAT_IP に変更します。

```
NODE_NAME ftsys9
NETWORK_INTERFACE lan1
    HEARTBEAT_IP      192.3.17.18
NETWORK_INTERFACE lan0
    HEARTBEAT_IP      15.13.170.18
NETWORK_INTERFACE lan3
NODE_NAME ftsys10
NETWORK_INTERFACE lan1
    HEARTBEAT_IP      192.3.17.19
NETWORK_INTERFACE lan0
    HEARTBEAT_IP      15.13.170.19
NETWORK_INTERFACE lan3
```

3. 新しい構成を検証します。

```
cmcheckconf -C clconfig.conf
```

4. 構成の変更を適用し、新しいバイナリ形式構成ファイルを全クラスターノードに配布します。

```
cmapplyconf -C clconfig.conf
```

データサブネットを構成しており、それをパッケージ構成に追加したい場合には、以下の手順を実行する必要があります。

1. パッケージを停止します。
2. パッケージ構成ファイルに新しいネットワーキング情報を追加します。
3. 従来のパッケージの場合には、必要に応じて、新しいネットワーキング情報をパッケージ制御スクリプトに追加します。
4. 新しいパッケージ構成を適用し、必要に応じて、制御スクリプトを再配布します。

詳細は、「稼働中のクラスターでのパッケージ再構成」(226 ページ) を参照してください。

例: パッケージで使われるサブネットの削除

この例では、サブネット 15.13.170.0 (lan0) を削除するとします。以下を行ってください。

1. このサブネットを使っているパッケージを停止し、対応するネットワーキング情報 (`monitored_subnet`、`ip_subnet`、`ip_address`。 [monitored_subnet \(169 ページ\)](#) 以降のこれらのパラメーターの説明を参照) を削除します。

詳細は、「稼働中のクラスターでのパッケージ再構成」(226 ページ) を参照してください。

2. `cmquerycl` を実行し、クラスター構成ファイルを取得します。

```
cmquerycl -c cluster1 -C clconfig.conf
```

3. ネットワークインターフェイス `lan0` と `lan3`、および他の影響を受けるノード上のネットワークインターフェイス(ある場合)をコメントアウトします。編集後のファイルのネットワーク関係の部分は、以下のとおりです。

```
NODE_NAME ftsys9
NETWORK_INTERFACE lan1
HEARTBEAT_IP 192.3.17.18
# NETWORK_INTERFACE lan0
# STATIONARY_IP 15.13.170.18
# NETWORK_INTERFACE lan3
NODE_NAME ftsys10
NETWORK_INTERFACE lan1
HEARTBEAT_IP 192.3.17.19
# NETWORK_INTERFACE lan0
# STATIONARY_IP 15.13.170.19
# NETWORK_INTERFACE lan3
```

4. 新しい構成を検証します。

```
cmcheckconf -C clconfig.conf
```

5. 構成の変更を適用し、新しいバイナリ形式構成ファイルを全クラスターノードに配布します。

```
cmapplyconf -C clconfig.conf
```

オンラインでのクラスターロック LUN 構成のアップデート

以下を行ってください。

- ① **重要:** 重要な情報については、「オンラインでクォーラム構成を変更したときの動作」(37 ページ) を参照してください。

1. クラスター構成ファイルで、各ノードの `CLUSTER_LOCK_LUN` の値を変更します。
2. `cmcheckconf` を実行して構成をチェックします。
3. `cmapplyconf` を実行して構成を適用します。

物理デバイスを交換する必要がある場合は、「ロック LUN の交換」(236 ページ) を参照してください。

MAX_CONFIGURED_PACKAGES の変更

Serviceguard A.11.18 から、クラスターの稼働中に MAX_CONFIGURED_PACKAGES を変更することができます。MAX_CONFIGURED_PACKAGES のデフォルトは、クラスター内で許されている最大数です。MAX_CONFIGURED_PACKAGES を変更するには、Serviceguard Manager を使うか、以下に示すように Serviceguard コマンドを使います。

クラスター内に存在する構成ファイルの現在のコピーを取得するには、次のように cmgetconf コマンドを使います。

```
cmgetconf -C <cluster_name> clconfig.conf
```

clconfig.conf ファイルを編集して、MAX_CONFIGURED_PACKAGES に新しい値を設定します。その後 cmcheckconf コマンドを使って新しい構成を検証します。-k オプションまたは -K オプションを指定すると、応答時間を大幅に短縮することができます。

cmapplyconf コマンドを使って構成の変更を適用し、新しい構成ファイルを全クラスターノードに送信します。-k オプションまたは -K オプションを指定すると、応答時間を大幅に短縮できます。

従来のパッケージの構成

- ❶ **重要:** 現在も、従来のパッケージを新規に作成することができます。Serviceguard NFS Toolkit などの Serviceguard Toolkit を使用している場合は、その製品のドキュメントを参照してください。

この項は、新しいパッケージを作成するためではなく、既存の従来のパッケージの保守や変更のためだけに使ってください。第6章「[パッケージとサービスの構成](#)」(157 ページ)で説明する方法は、新しいパッケージを作成するための簡単で効率のよい方法です。この方法では、パッケージは小さなモジュールから構築でき、独立したパッケージ制御スクリプトが不要で、手作業で配布する必要もありません。

従来のパッケージをモジュラーパッケージに変換する場合は、「[従来のパッケージからモジュラーパッケージへの移行](#)」(226 ページ)を参照してください。Serviceguard Toolkit パッケージは変換しないでください。

従来のパッケージの作成/変更には、大きく以下の手順が必要です。

1. パッケージ構成ファイルの作成
2. パッケージ構成ファイルの編集
3. パッケージ制御スクリプトの作成
4. パッケージ制御スクリプトの編集
5. 制御スクリプトのクラスターノードへの配布
6. パッケージ構成ファイルの適用

これらの作業のそれぞれについて、以降の項で説明します。

従来のパッケージ構成の作成

パッケージ構成プロセスでは、クラスターノードでパッケージを起動するときに、パッケージマネージャーにより起動される一連のアプリケーションサービスを定義します。また、パッケージ構成には、パッケージを実行できるクラスターノードの優先順位リストと、パッケージで可能なフェイルオーバーのタイプの定義が含まれます。

Serviceguard Manager を使ったパッケージの構成

Serviceguard Manager を使って、従来のパッケージとその制御スクリプトを作成することができます。詳細は、ヘルプを参照してください。

Serviceguard コマンドを使ったパッケージの構成

以下の手順に従って、従来のパッケージを作成します。

1. 構成するパッケージごとに、`$SGCONF` ディレクトリ内にサブディレクトリを作成します。

```
mkdir $SGCONF/pkg1
```


ディレクトリ名は自由に指定できます。(使用しているバージョンの Linux における Serviceguard ディレクトリについては、「[Serviceguard のファイルの位置](#)」(122 ページ)を参照してください。)
2. たとえば、次のコマンドを使って、パッケージごとにパッケージ構成ファイルを作成します。

```
cmmakepkg -p $SGCONF/pkg1/pkg1.conf
```


構成ファイルにはファイル名を自由に指定できます。
3. 各構成ファイルを編集し、各パッケージのパッケージ名、ノード優先順位リスト (名前は 39 バイト以下)、制御スクリプトの場所、フェイルオーバーパラメーターを指定します。これらの情報は、パッケージ構成ワークシートに記録されています。

パッケージの構成手順

以下の手順に従ってフェイルオーバーパッケージを構成します。

1. ボリュームグループとマウントポイントだけを構成します。
2. 制御スクリプトを全ノードに配布します。
3. 構成を適用します。
4. パッケージを起動して、ノード間でパッケージが移動できることを確かめます。
5. パッケージを停止します。
6. パッケージの IP アドレスとアプリケーションサービスを制御スクリプトに定義します。
7. 制御スクリプトを全ノードに配布します。
8. パッケージを起動して、アプリケーションが正しく動作することと、サービス中断時のパッケージのフェイルオーバーが正しく行われることを確かめます。

パッケージ構成ファイルの編集

「[Serviceguard コマンドを使ったパッケージの構成](#)」(218 ページ)の手順 2 で生成したファイルを編集します。以下の個条書きをチェックリストとして使います。

- `PACKAGE_TYPE`: パッケージタイプを指定します。「[パッケージのタイプ: フェイルオーバー、マルチノード、システムマルチノード](#)」(158 ページ)と「`package_type`」(163 ページ)を参照してください。

注記: モジュラーパッケージでは、パッケージ構成ファイル内のパラメーター名とリテラル値のデフォルトの形式は小文字です。従来のパッケージではデフォルトは大文字です。互換性に関する問題はありません。Serviceguard は、少なくともパラメーター名については、大文字と小文字を区別しないためです。

この項は、既存の従来のパッケージを再構成する際に使用することを主目的に記述しているため、連続性を持たせ、従来のパラメーター名(大文字)を使います。ただし、`cmmakepkg` や `cmgetconf` を使って構成ファイルを作成すると、モジュラーパッケージで使われるパラメーター名で作成されます。名前を変更する方法については、下記の注記と「[パッケージのパラメーターの説明](#)」(162 ページ)を参照してください。

- `FAILOVER_POLICY`: フェイルオーバーパッケージの場合には、`failover_policy` (166 ページ)を指定します。
- `FAILBACK_POLICY`: フェイルオーバーパッケージの場合には、`failback_policy` (167 ページ)を指定します。
- `NODE_NAME`: パッケージを実行できるノードを指定します。`node_name`(164 ページ)で説明しています。

- `AUTO_RUN`: パッケージを自動的に起動するか、手作業で起動するかを指定します。
[auto_run \(164 ページ\)](#) で説明しています。
 - `NODE_FAIL_FAST_ENABLED`: 方針を指定します。[node_fail_fast_enabled \(164 ページ\)](#) で説明しています。
 - `RUN_SCRIPT` および `HALT_SCRIPT`: パッケージ制御スクリプトのパス名を指定します (次項で説明)。デフォルトはありません。ファイルとディレクトリのパーミッションには、`rwxr-xr-x` または `r-xr-xr-x` (755 または 555) を設定します。
(スクリプトのタイムアウト): [run_script_timeout \(165 ページ\)](#) と [halt_script_timeout \(165 ページ\)](#) を指定します。
`SCRIPT_LOG_FILE` (オプション): `RUN_SCRIPT` と `HALT_SCRIPT` でメッセージが記録されるファイルの絶対パス名を指定します。このパスを指定しなかった場合には、Serviceguard は、各スクリプトパスの後に「.log」を付けてファイル名とし、そのファイルにメッセージを記録します。
-
- パッケージに再配置可能 IP アドレスが割り当てられていて、そのパッケージを監視する場合には、`SUBNET` を指定します (こうすると、サブネット障害が発生した場合には、パッケージは停止します)。
これはクラスター構成ですでに指定されているサブネットである必要があり、IPv4 サブネットと IPv6 サブネットのどちらでも構いません。リンクローカルサブネットであってはなりません (リンクローカルパッケージ IP は許されていません)。[monitored_subnet \(169 ページ\)](#) を参照してください。
-
- ① **重要:** ここで指定した各サブネットは、`NETWORK_INTERFACE` パラメーター、および `HEARTBEAT_IP` または `STATIONARY_IP` のいずれかのパラメーターを用いてクラスター構成ファイルに指定してある必要があります。詳細は、「[クラスター構成のパラメーター](#)」(82 ページ) を参照してください。
「[定常 IP アドレスおよび再配置可能 IP アドレスと監視対象サブネット](#)」(54 ページ) と [monitored_subnet \(169 ページ\)](#) も参照してください。
- 重要:** クロスサブネット構成については、「[クロスサブネットフェイルオーバーの構成](#)」(224 ページ) を参照してください。
-
- パッケージでサービスを起動する場合には、[service_name \(171 ページ\)](#) で説明されている `SERVICE_NAME` と、[service_fail_fast_enabled \(172 ページ\)](#) で説明されている `SERVICE_FAIL_FAST_ENABLED` と [service_halt_timeout \(172 ページ\)](#) で説明されている `SERVICE_HALT_TIMEOUT` の値を指定します。各サービスに対してこれらの 3 つのパラメーターを指定します。
-
- ① **重要:** 有効な `SERVICE_NAME` の規則は、Serviceguard A.11.18 ではより厳しい制限になっていることに注意してください。
-
- `ACCESS_CONTROL_POLICY`: 非 root ユーザーに対して、このパッケージ用の `PACKAGE_ADMIN` 権限を与えることができます。
詳細は、`user_name`、`user_host`、`user_role` [user_name \(178 ページ\)](#) の項目と、「[クラスターへのアクセスの制御](#)」(146 ページ) を参照してください。
 - パッケージが別のパッケージに依存する場合には、`DEPENDENCY_NAME`、`DEPENDENCY_CONDITION`、`DEPENDENCY_LOCATION` に値を設定します。
詳細は、以降の該当するパラメーターの説明([167 ページ](#))と、「[パッケージの依存関係について](#)」([100 ページ](#)) を参照してください。

パッケージ制御スクリプトの作成

従来のパッケージでは、パッケージ制御スクリプトには、パッケージに含まれる全サービスの実行、動作の監視、障害への対応、パッケージの停止(必要な場合)に必要なすべての情報が含まれています。Serviceguard Manager、Serviceguard コマンド、その両方の組み合わせを使って、パッケージ制御スクリプトを作成または変更することができます。

各パッケージには、それぞれ専用のパッケージ制御スクリプトが必要で、このスクリプトは実行可能プログラムでなければなりません。

セキュリティ上の理由から、制御スクリプトは、パス名に `cmcluster` が含まれるディレクトリに置かなければなりません。このスクリプトをパッケージと同じディレクトリに置いて、パッケージ構成ファイル内の `RUN_SCRIPT` および `HALT_SCRIPT` パラメーターで指定したスクリプト名と同じ名前にしておきます。パッケージ制御スクリプトのテンプレートには、パッケージの実行命令と停止命令の両方が含まれています。1つのスクリプトで実行命令と停止命令の両方を行うことができます。また、個別のスクリプトで行うこともできます。

`cmmakepkg` を使って制御スクリプトを作成した後、その制御スクリプトを編集します。サンプルのフェイルオーバーパッケージ `pkg1` 用のテンプレートを作成するには、次のようにします。

まず、制御スクリプトのテンプレートを生成します。

```
cmmakepkg -s $SGCONF/pkg1/pkg1.sh
```

次に、スクリプトをカスタマイズします。「[パッケージ制御スクリプトのカスタマイズ](#)」を参照してください。

パッケージ制御スクリプトのカスタマイズ

以下のようにカスタマイズする必要があります。詳細は、「[パッケージのパラメーターの説明](#)」(162 ページ)の関連エントリを参照してください。

- `PATH` 文をアップデートして、サービスの起動に必要なすべてのパスを指定します。
- 必要に応じてリモートデータレプリケーション方法やソフトウェア RAID データレプリケーション方法を指定します。

△ **注意:** XDC や CLX の製品を使用していない場合は、`REMOTE DATA REPLICATION DEFINITION` セクションは変更しないでください。XDC や CLX の製品を使用する場合は、製品のドキュメントを参照してください。

- LVM を使用している場合、`VG[]` 配列パラメーターを使用して、アクティブ化するボリュームグループの名前を入力します。また必要に応じて、ファイルシステムのマウントおよびマウント解除のオプション等の、ストレージアクティブ化コマンドの適切なオプションを選択します。ファイルシステムタイプを指定します(RHEL 5 では `ext3` がデフォルトです。RHEL 6 では `ext4` がデフォルトです。`reiserfs` または `gfs` も指定可能です。詳細は、[fs_mount_retry_count](#) (175 ページ) 以降の `fs_` パラメーターの説明を参照してください)。

注記: Red Hat GFS と `reiserfs` は、Serviceguard A.11.20.00 ではサポートされません。

- 論理ボリュームの名前と、それらにマウントされるファイルシステムの名前を追加します。
- ファイルシステムのマウントとマウント解除の再試行オプションを指定します。
- パッケージが、多数のボリュームグループやディスクグループを使うか、多数のファイルシステムをマウントする場合は、`vgchange`、`mount/umount`、`fsck` の同時操作数を大きくすることを検討してください。
- パッケージの IP サブネットと IP アドレスのペアを定義します。IPv4 アドレスまたは IPv6 アドレスを使うことができます。

- サービス名を追加します。
- サービスコマンドを追加します。
- 必要に応じて、サービスの再起動パラメーターを追加します。
サービスについての詳細は、[service_name\(171 ページ\)](#)以降の `service_` で始まるパラメーターの説明を参照してください。

パッケージ制御スクリプトへのユーザー定義関数の追加

パッケージ制御スクリプトには、パッケージ起動時/停止時に実行するシェルコマンドを追加できます。これらのコマンドは、スクリプトの CUSTOMER DEFINED FUNCTIONS セクションに追加します。

パッケージ化されたアプリケーションの初期化や停止を行うコマンドのように、パッケージで短時間だけ実行するプロセスを実行する必要がある場合には、これらのプロセスも CUSTOMER DEFINED FUNCTIONS から起動できます。

また、CUSTOMER DEFINED FUNCTIONS を使って、パッケージがシャットダウンした原因を調べることができます。「[パッケージがシャットダウンした原因の調査](#)」(117 ページ)を参照してください。

コマンドを追加したスクリプトの例を以下に示します。この例では、`date` と `echo` というコマンドを追加して、ファイルにパッケージ起動時/停止時のログを記録します。

```
# START OF CUSTOMER DEFINED FUNCTIONS

# This function is a place holder for customer defined functions.
# You should define all actions you want to happen here, before the service is
# started.  You can create as many functions as you need.

function customer_defined_run_cmds
{
# ADD customer defined run commands.
: # do nothing instruction, because a function must contain some command.
date >> /tmp/pkg1.datelog
echo 'Starting pkg1' >> /tmp/pkg1.datelog
test_return 51
}

# This function is a place holder for customer defined functions.
# You should define all actions you want to happen here, before the service is
# halted.

function customer_defined_halt_cmds
{
# ADD customer defined halt commands.
: # do nothing instruction, because a function must contain some command.
date >> /tmp/pkg1.datelog
echo 'Halting pkg1' >> /tmp/pkg1.datelog
test_return 52
}

# END OF CUSTOMER DEFINED FUNCTIONS
```

ユーザー定義関数への Serviceguard コマンドの追加

パッケージ制御スクリプトの CUSTOMER DEFINED FUNCTIONS セクションには、`cmmodpkg` などの Serviceguard コマンドを追加できます。ただし追加できるのは、パッケージそのものとは直接データをやり取りしないコマンドだけです。

Serviceguard コマンドが他のパッケージとデータをやり取りする場合は、コマンドループが発生しないように注意してください。コマンドループが発生するのは次のような場合です。たとえば `pkg1` が `pkg2` に対して `cmmodpkg -d` を実行し、`pkg2` が `pkg1` に対して `cmmodpkg -d` を実行するとします。`pkg1` と `pkg2` の両方のパッケージが同時に起動されると、`pkg1` は `pkg2` に対して `cmmodpkg` を実行しようとして、ところがこの `cmmodpkg` コマンドは、`pkg2` の起動が完了しなければ実行できません。一方 `pkg2` も `pkg1` に対して `cmmodpkg` を実行し

ようにしますが、同様に、pkg2 は pkg1 の起動が完了しなければ実行できないので、コマンドループが発生します。

このようなコマンドループが発生しないように、すべてのパッケージに `RUN_SCRIPT_TIMEOUT` と `HALT_SCRIPT_TIMEOUT` を指定しておくといでしょう。制御スクリプトに `Serviceguard` コマンドが含まれているパッケージには、この方法は特に有効です。タイムアウトの指定をしていないときに上記のようなコマンドループが発生すると、クラスターがハングするなど、望ましくない状況が発生する可能性があります。

追加製品のサポート

パッケージ制御スクリプトのテンプレートには、`Serviceguard Extended Distance Cluster (XDC) for Linux` などの追加製品との共用のための仕組みがあります。制御スクリプト内に用意されているフックを使ってパッケージを作成する方法についての詳細は、追加製品のドキュメントを参照してください。

パッケージ構成の確認

`Serviceguard` は、作成される構成をチェックしてエラーを報告します。

従来のパッケージでは、`Serviceguard Manager` を使ってこの操作を行うことができます。すなわち、[チェック] をクリックし、パッケージ構成タブで指定したパッケージ構成の検証や、制御スクリプトに対して行った変更の確認を行います。パッケージ全体を確認するには、[適用] をクリックします。詳細は、ローカルのヘルプを参照してください。

コマンド行を使っている場合は、次のコマンドを実行して、作成したパッケージ構成ファイルを検証します。

```
cmcheckconf -v -P $SGCONF/pkg1/pkg1.conf
```

エラーは、標準出力に表示されます。必要であればエラーを訂正するためにファイルを編集してから、コマンドを再度実行します。エラーがなくなれば完了です。

以下の項目がチェックされます (`Serviceguard Manager` と `cmcheckconf` コマンドのどちらを使っても同じです)。

- パッケージ名が有効で、少なくとも 1 つの `NODE_NAME` エントリーが含まれている。
- パラメーターエントリーが重複していない。
- パラメーターの値が許容範囲内である。
- 実行および停止スクリプトはクラスター内のすべてのノードに存在し、実行可能プログラムである。
- 実行および停止スクリプトのタイムアウト時間が 4294 秒未満である。
- 構成されたリソースはクラスターノード上で利用可能である。
- 依存関係が構成されている場合には、依存される側のパッケージがクラスター内で構成済みである。

構成の配布

`Serviceguard Manager` または `Linux` コマンドを使用して、バイナリ形式クラスター構成ファイルをクラスター内の各ノードに配布します。

Serviceguard Manager による構成と制御スクリプトの配布

`Serviceguard Manager` でのパッケージの作成を終えたら、[構成の適用] をクリックします。パッケージ構成にエラーがなければ、バイナリファイルに変換され、クラスターの各ノードに配布されます。

Linux コマンドによるパッケージ制御スクリプトのコピー

- ① **重要:** クロスサブネット構成では、パッケージで再配置可能 IP アドレスを使っている場合、すべてのノードで同じパッケージ制御スクリプトを使うことはできません。「[クロスサブネットフェイルオーバーの構成](#)」(224 ページ) を参照してください。

Linux コマンドを使用して、ファイルを作成したノードから、そのパッケージの実行が可能な全ノード上の同一パス名へ、パッケージ制御スクリプトをコピーします。コピーには、使い慣れたファイル転送機能 (たとえば `scp` や `ftp` など) を使用します。たとえば次のように、`ftsys9` から `scp` コマンドを使用して、`ftsys10` へパッケージ制御スクリプトをコピーできます。

```
scp $SGCONF/pkg1/control.sh ftsys10:$SGCONF/pkg1/control.sh
```

Linux コマンドによるバイナリ形式クラスター構成ファイルの配布

クラスター構成ファイルとパッケージ構成ファイルを作成したノード上で、以下の手順を行います。

- 次のコマンドを使用して、構成ファイルの内容が正しいことを確かめます。次のコマンドを使用します。

```
cmcheckconf -C $SGCONF/cmcl.conf -P $SGCONF/pkg1/pkg1.conf
```

- バイナリ形式構成ファイルを生成して、ノード全体に配布します。

```
cmapplyconf -v -C $SGCONF/cmcl.conf -P $SGCONF/pkg1/pkg1.conf
```

`cmapplyconf` コマンドにより、バイナリのクラスター構成ファイルが生成され、クラスター内の全ノードへ配布されます。これにより、クラスター内の全ノードが同じ内容の構成ファイルを持つことができます。

注記: クラスターまたはパッケージの構成ファイルの内容を変更した後は、必ず `cmcheckconf` と `cmapplyconf` を実行しなければなりません。

クロスサブネットフェイルオーバーの構成

サブネットにまたがってフェイルオーバーするように従来のパッケージを構成するためには (「[クロスサブネット構成](#)」(25 ページ) を参照)、追加の構成作業が必要です。

パッケージ `pkg1` を、`NodeA`、`NodeB`、`NodeC`、および `NodeD` からなるクラスター内のすべてのノード間でフェイルオーバーできるように構成するとします。

`NodeA` と `NodeB` は、`NodeC` と `NodeD` が使わないサブネット `15.244.65.0` を使います。

`NodeC` と `NodeD` は、`NodeA` と `NodeB` が使わないサブネット `15.244.56.0` を使います。

(`cmquerycl` の出力例は、「[クロスサブネット情報の取得](#)」(144 ページ) を参照してください。)

node_name の構成

まず、`pkg1` が他のサブネット上のノードにフェイルオーバーするのは、やむを得ない場合だけであることを確認する必要があります。たとえば、このパッケージが `NodeA` 上で実行されていて、フェイルオーバーする必要がある場合、クロスサブネットのオーバーヘッドがある `NodeC` や `NodeD` へのフェイルオーバーを行うのは、同じサブネット上の `NodeB` へのフェイルオーバーを試みた後です。

`nodeA` が `pkg1` のプライマリノード (そのパッケージが通常起動されるノード) だとすると、パッケージ構成ファイルに次のような `node_name` エントリーを作成します。

```
node_name nodeA
node_name nodeB
node_name nodeC
node_name nodeD
```

monitored_subnet_access の構成

サブネット15.244.65.0または15.244.56.0を監視するには、pkg1のパッケージ構成ファイルの `monitored_subnet` と `monitored_subnet_access` を、次のように構成します。

```
monitored_subnet 15.244.65.0
monitored_subnet_access PARTIAL
monitored_subnet 15.244.56.0
monitored_subnet_access PARTIAL
```

注記: いずれのサブネットもすべてのノード上で利用可能というわけではないため、これらのサブネットのいずれかに対して `monitored_subnet_access` をFULLとして構成する(または、`monitored_subnet_access` を設定しない) と、パッケージの構成が失敗します。

サブネット固有のパッケージ制御スクリプトの作成

次に、パッケージが4つのノード上で動作するように制御スクリプトを作成する必要があります。

- ① **重要:** クロスサブネット構成では、再配置可能 IP アドレスを使っている場合、異なるサブネット上の複数のノードで、1つのパッケージ制御スクリプトを共有することはできません。この場合、各サブネット上のノードで使う個別の制御スクリプトを作成する必要があります。

この例では、pkg1のパッケージ制御スクリプトのコピーを2つ作成し、以下のようにサブネット15.244.65.0または15.244.56.0用にカスタマイズするエントリーを追加し、各ノードに該当のスクリプトをコピーします。

nodeA と nodeB の制御スクリプトのエントリー

```
IP[0] = 15.244.65.82
SUBNET[0] 15.244.65.0
IP[1] = 15.244.65.83
SUBNET[1] 15.244.65.0
```

nodeC と nodeD の制御スクリプトのエントリー

```
IP[0] = 15.244.56.100
SUBNET[0] = 15.244.56.0
IP[1] = 15.244.56.101
SUBNET[1] = 15.244.56.0
```

パッケージの再構成

パッケージを最初に構成したときとほぼ同じ方法で、パッケージを再構成します。モジュールパッケージの場合は、[第6章「パッケージとサービスの構成」](#) (157ページ)を参照してください。古いパッケージの場合は、「[従来のパッケージの構成](#)」 (218ページ)を参照してください。

パッケージの再構成は、クラスターの停止中でも稼働中でも行えます。一部のケースでは、パッケージそのものが稼働中でも可能です。可能な変更の種類や、変更が適用される時期は、パッケージが稼働中かどうかにより異なります。

動作中のパッケージを再構成すると、`cmapplyconf` が成功していても、後でパッケージで障害が発生する場合があります。

たとえば、2つのボリュームグループを持つパッケージを考えます。このパッケージを起動すると、両方のボリュームグループがアクティブ化されます。このパッケージの動作中に構成ファイルを変更して、1つのボリュームグループしか記述しないようにしても、`cmapplyconf` コマンドは正常終了します。しかし、`cmhaltpkg` コマンドを実行すると、停止は失敗します。変更されたパッケージの現在の構成ファイル内には1つのボリュームグループしか記述されて

いないために、起動時にアクティブ化した両方のボリュームグループが非アクティブ化できないからです。

詳細は、「構成変更が可能なパッケージの状態」(228 ページ) を参照してください。

従来のパッケージからモジュラーパッケージへの移行

Serviceguard のコマンド `cmigratepkg` を使うと、従来のパッケージからモジュラーパッケージへの移行処理が、可能な限り自動化されます。この方法で、すべてではありませんが多くのパッケージを移行できます。詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard] -> [White Papers] にあるホワイトペーパー『Package Migration from Legacy Style to Modular Style』を参照してください。

Serviceguard Toolkit パッケージは変換しないでください。

注記: `cmigratepkg` コマンドを実行するシステムには Perl5.8.3 以降が必要です。

稼働中のクラスターでのパッケージ再構成

クラスター稼働中でもパッケージは再構成できます。場合によっては、稼働しているパッケージそのものを再構成することもできます。「構成変更が可能なパッケージの状態」(228 ページ) を参照してください。この作業を行うには、Serviceguard Manager (従来のパッケージの場合) を使うか、Serviceguard コマンドを使います。

Serviceguard コマンドでパッケージを変更するには、以下の手順を実行します (`pkg1` はパッケージ例)。

1. 必要な場合は、以下のコマンドでパッケージを停止します。

```
cmhaltpkg pkg1
```

See このステップが必要かどうかは、「構成変更が可能なパッケージの状態」を参照して判断してください。

2. まだ手元にない場合は、パッケージ名を指定して `cmgetconf` コマンドを実行し、パッケージの構成ファイルを取得します。

```
cmgetconf -p pkg1 pkg1.conf
```

3. パッケージ構成ファイルを編集します。

- ① **重要:** A.11.18 では、パッケージ名、依存関係名、サービス名についての制限が一層厳しくなりました。新しい規則(「[package_name](#)」(163 ページ) を参照) に準拠していない名前のパッケージでも実行することはできますが、これらのパッケージを再構成する場合には、準拠していない名前を変更する必要があります。`cmcheckconf` と `cmapplyconf` では、新しい規則が適用されます。

4. 次のコマンドを使用して、変更をチェックします。

```
cmcheckconf -v -P pkg1.conf
```

5. 変更した構成を全ノードに配布します。

```
cmapplyconf -v -P pkg1.conf
```

6. 従来のパッケージの場合には、パッケージを実行する全ノードにパッケージ制御スクリプトをコピーします。

稼働中のパッケージで使用される外部スクリプトの名前変更または交換

多くの場合、外部スクリプトを使用するパッケージの稼働中に、その外部スクリプト(178 ページ) の名前を変更できますが、注意が必要です。以下の指示に従ってください。

1. 古いスクリプトのコピーを作成し、新しい名前で保存します。必要な場合はコピーを編集します。
2. パッケージ構成ファイルを編集して、新しい名前を使用できるようにします。

3. そのパッケージ用に構成されているすべてのノードに新しいスクリプトを配布します。
新しいスクリプトを、適切なファイルモードと所有権を設定して正しいディレクトリに配置していることを確認します。
4. `cmcheckconf` を実行して、新しい外部スクリプトでのパッケージ構成を検証します。

△ **注意:** `cmcheckconf` が失敗する場合は、エラーをすべて修正してから、次に進んでください。

5. 稼働中のパッケージに対して `cmapplyconf` を実行します。
これにより、元のスクリプトで起動されたりリソースが停止し、その後、新しいスクリプトが必要とするリソースが起動します。
6. ここまでの手順が完了すれば、パッケージを実行するように構成されているすべてのノードで、元の外部スクリプトを削除しても問題ありません。

停止しているクラスターでのパッケージの再構成

クラスターが停止している場合でも、パッケージの構成を恒久的に変更できます。「稼働中のクラスターでのパッケージ再構成」と同じ手順を実行します。

稼働中のクラスターへのパッケージの追加

新しいパッケージを作成して、クラスターや他のパッケージが稼働している間にそのパッケージをクラスターに追加できます。追加できるパッケージの数は、クラスター構成ファイル中の `MAX_CONFIGURED_PACKAGES` の値により異なります。

パッケージは、第6章「パッケージとサービスの構成」(157 ページ)の章で説明した手順で作成します。次に、次のようなコマンドを使って、新しく作成した `pkg1` の構成を稼働中のクラスター上で検証します。

```
cmcheckconf -P $SGCONF/pkg1/pkg1conf.conf
```

次のようなコマンドを使って、新しいパッケージ構成をクラスター内の全ノードに配布します。

```
cmapplyconf -P $SGCONF/pkg1/pkg1conf.conf
```

従来のパッケージの場合には、そのパッケージを実行する全ノードのディレクトリ `$SGCONF/pkg1` に制御スクリプトをコピーしてください。

稼働中のクラスターからのパッケージの削除

Serviceguard では、他のパッケージが依存しているパッケージを削除することはできません。依存関係を確認するには、`cmviewcl -v -l <package>` を使います。システムマルチノードパッケージは、動作中のクラスターからは削除できません。

Serviceguard Manager を使ってパッケージを削除することができます。

Serviceguard コマンド行では、ほとんどの場合、`cmdeleteconf` コマンドを使って、クラスター内の全ノードからパッケージを削除できます。これらのコマンドを実行すると、クラスター内の全ノードのバイナリ構成ファイルからそのパッケージに関する情報が削除されます。これらのコマンドは、パッケージが停止しているときしか実行できません。クラスターは稼働中でも構いません。

以下の例では、フェイルオーバーパッケージ `mypkg` を停止させた後、クラスターからパッケージ構成を削除します。

```
cmhaltpkg mypkg cmdeleteconf -p mypkg
```

コマンドを実行すると、ファイルの削除を確認するプロンプトが表示されます。ただし `-f` オプションを指定した場合は、プロンプトは表示されません。またコマンドを実行しても、ディレクトリ `$SGCONF/mypkg` は削除されません。

サービスの再起動カウンターのリセット

サービスの再起動カウンターでは、パッケージサービスが自動的に再起動された回数をカウントします。この値により、パッケージサービスを自動的に再起動できる最大回数を超えた時期が判別されます。

数回の試行後、パッケージサービスの再起動に成功した場合、パッケージマネージャーは再起動カウンターを自動的にリセットしません。次のように `cmmodpkg -R -s myservice pkg1` コマンドを使って、カウンターをオンラインでリセットすることができます。

```
cmmodpkg -R -s myservice pkg1
```

これによりカウンターがゼロに戻ります。再起動カウンターの現在の値は、`cmviewcl -v` の出力に表示されます。

構成変更が可能なパッケージの状態

多くの場合、パッケージの構成は、パッケージを動作させたままで変更できます。次の表には、例外 (パッケージが動作してはならない、または動作していると、結果が期待どおりにならない) を示します。また、モジュラーパッケージと従来のパッケージとの違いも示しています。

△ **注意:** パッケージの稼働中は、パッケージ構成の変更には十分注意してください。

パッケージをオンラインで再構成する (パッケージ自体が稼働している間にパッケージに対して `cmapplyconf` を実行する) と、`cmapplyconf` が成功して変更エラーが含まれていないことが確認されても、その後で、パッケージが停止することがあります。

たとえば、パッケージの稼働中にファイルシステムをパッケージに追加すると、`cmapplyconf` はさまざまなチェックを行い、ファイルシステムおよびそのマウントポイントが存在するかどうかを確認します。ただし、ファイルシステムのチェックやファイルシステムのマウントは実際は `cmapplyconf` が成功した後でしか実行できず、稼働中のパッケージでこれらのタスクのいずれかが失敗すると、パッケージ全体が停止します。

もう1つの例として、外部スクリプトを使用するパッケージの稼働中におけるそのスクリプトの名前変更、変更、または交換があります。パッケージが、そのスクリプトが管理するリソースに依存している場合、スクリプトを交換するとパッケージが停止します。「稼働中のパッケージで使用される外部スクリプトの名前変更または交換」(226 ページ) を参照してください。

一般に、従来のパッケージよりもモジュラーパッケージの方が、オンライン変更に対応する範囲が広がります。ただし、一部のケースでは、モジュラーパッケージの機能にできるだけ匹敵するように従来のパッケージの機能がアップグレードされています。これらのケースを表に示します。従来のパッケージとモジュラーパッケージについての詳細は、第6章(157 ページ) を参照してください。

注記: 「パッケージに対する変更」で**従来**または**モジュラー**のいずれも付記されていない場合、「変更が可能なパッケージの状態」が両方のタイプのパッケージに適用されます。変更が可能でも、HP が推奨しないものは「稼働していないことが望ましい」としています。

① **重要:** この表にリストされていないアクションは、両方のタイプのパッケージについて、パッケージを動作させたままで実行できます。

クラスターは動作させたままにでき、再構成するパッケージ以外のパッケージも動作させたままにしておくことができます。パッケージ構成**ファイル**はいつでも変更できます。ただし、表に示されているとおり、動作中のパッケージに対して `cmapplyconf` コマンドや Serviceguard Manager を使って適用することはできない場合があります。

注記: パッケージ構成を変更するときには、クラスター内の全ノードの電源が入っていて、ノードにアクセスできる状態でなければなりません。

表 12 パッケージに対する変更の種類

パッケージに対する変更	変更が可能なパッケージの状態
パッケージの削除	パッケージが稼働してはならない。 注記: 他のパッケージが依存しているパッケージは削除できません。
パッケージタイプの変更	パッケージが稼働してはならない。
モジュールの追加または削除 (モジュラーパッケージ)	パッケージは稼働していてもよい。
実行スクリプトの内容の変更 (従来のパッケージ)	パッケージは稼働していてもよいが、起動最中でないことが望ましい。 パッケージ起動最中にそのパッケージのスクリプトを編集すると、タイミングによっては障害が発生するおそれがある。
停止スクリプトの内容の変更 (従来のパッケージ)	パッケージは稼働していてもよいが、停止最中でないことが望ましい。 パッケージ停止最中にそのパッケージのスクリプトを編集すると、タイミングによっては障害が発生するおそれがある。
サービスの追加または削除 (モジュラーパッケージ)	パッケージは稼働していてもよい。 Serviceguard は、 <code>service_name</code> や <code>service_cmd</code> へのいずれの変更も既存のサービスの削除および新しいサービスの追加として処理します。つまり既存のサービスは停止されるということです。
サービスの追加または削除 (従来のパッケージ)	パッケージが稼働してはならない。
<code>service_restart</code> の変更 (モジュラーパッケージ)	パッケージは稼働していてもよい。 Serviceguard は、新しい値が現在の再起動数を下回る場合には、変更を許しません。(必要な場合は、 <code>cmmodpkg -R<service_name> <package></code> を使用して再起動数をリセットしてください。)
<code>SERVICE_RESTART</code> の変更 (従来のパッケージ)	パッケージが稼働してはならない。
<code>SUBNET</code> (制御スクリプト内)の追加または削除 (従来のパッケージ)	パッケージが稼働してはならない。(クロスサブネット構成にも適用) パッケージが稼働してはならない。サブネットはすでにクラスターに構成されている必要があります。
<code>ip_subnet</code> の追加または削除 (モジュラーパッケージ)	パッケージは稼働していてもよい。 重要な情報については、「 ip_subnet 」(170 ページ) を参照してください。 Serviceguard は、パッケージの <code>node_name</code> リスト上のすべてのノードに対して構成されていない <code>ip_subnet</code> を追加しようとすると、変更を拒否します。
<code>ip_address</code> の追加または削除 (モジュラーパッケージ)	パッケージは稼働していてもよい。 重要な情報は、「 ip_subnet 」(170 ページ) および「 ip_address 」(171 ページ) を参照してください。Serviceguard は、指定した <code>ip_subnet</code> 上に構成できない <code>ip_address</code> 、あるいはパッケージの <code>node_name</code> リスト上のすべてのノードに対して構成されていないサブネット上にある <code>ip_address</code> を追加しようとすると、変更を拒否します。
<code>IP</code> (制御スクリプト内)の追加または削除 (従来のパッケージ)	パッケージが稼働してはならない。(クロスサブネット構成にも適用)
パッケージの <code>ip_subnet_node</code> リスト (171 ページ) (クロスサブネット構成内) のノードの追加または削除	パッケージは稼働していてもよい。 Serviceguard は、指定した <code>ip_subnet</code> が構成されていないノードを追加しようとすると、変更を拒否します。
サブネットの監視の追加または削除 (モジュラーパッケージ) の場合には	パッケージは稼働していてもよい。 Serviceguard は、追加するサブネットがダウンしている場合には、稼働しているパッケージが異常終了するおそれがあるため、変更を許しません。

表 12 パッケージに対する変更の種類 (続き)

パッケージに対する変更	変更が可能なパッケージの状態
<i>monitored_subnet</i> または従来のパッケージの場合にはパッケージ構成ファイルの <i>SUBNET</i>)	
<i>pv</i> の追加、変更、または削除: モジュラーパッケージ	パッケージが稼働してはならない。 注記: <i>pv</i> (177 ページ) は、HP パートナーが専用に使用するものです。
ボリュームグループの追加 (モジュラーパッケージ)	パッケージは稼働していてもよい。
ボリュームグループの追加 (従来のパッケージ)	パッケージが稼働してはならない。
ボリュームグループの削除 (モジュラーパッケージ)	パッケージは稼働していないことが望ましい。 注意: ファイルシステム、アプリケーション、またはその他のリソースがボリュームグループを使用している場合、そのボリュームグループを削除すると、問題が発生する可能性があります。また、「Remove a file system: modular package 」の下での CAUTION は、ボリュームグループを使用するすべてのファイルシステムに該当します。
ボリュームグループの削除 (従来のパッケージ)	パッケージが稼働してはならない。
ファイルシステムの変更 (モジュラーパッケージ)	パッケージは稼働していないことが望ましい (<i>fs_umount_opt</i> のみを変更しているのではない限り)。 <i>fs_umount_opt</i> 以外のファイルシステムのオプションを変更すると、ファイルシステムをマウント解除し (既存の <i>fs_umount_opt</i> を使用)、その後、新しいオプションで再マウントしなければならないため、問題を引き起こすおそれがあります。「ファイルシステムの削除 (モジュラーパッケージ)」での「注意」がこのケースにも当てはまります。 <i>fs_umount_opt</i> のみを変更した場合、ファイルシステムはマウント解除されません。新しいオプションは、パッケージが停止したとき、あるいはファイルシステムが他の何らかの理由でマウント解除されたときに有効になります。
ファイルシステムの追加 (モジュラーパッケージ)	パッケージは稼働していてもよい。
ファイルシステムの追加または変更 (従来のパッケージ)	パッケージが稼働してはならない。
ファイルシステムの削除 (モジュラーパッケージ)	パッケージは稼働していないことが望ましい。 注意: ファイルシステムをマウント解除できない場合は、これが実行プロセスによって使用中であるため、ファイルシステムを削除すると問題が生じる場合があります。この場合は、Serviceguard はプロセスを抹消します。このためパッケージが異常終了する可能性があります。
ファイルシステムの削除 (従来のパッケージ)	パッケージが稼働してはならない。
<i>concurrent_fsck_operations</i> 、 <i>fs_mount_retry_count</i> 、 <i>fs_umount_retry_count</i> の変更 (モジュラーパッケージ)	パッケージは稼働していてもよい。 これらの変更そのものでは、いずれのファイルシステムもマウント解除されることはありません。
<i>concurrent_fsck_operations</i> 、 <i>fs_mount_retry_count</i> 、 <i>fs_umount_retry_count</i> の変更 (従来のパッケージ)	パッケージが稼働してはならない。

表 12 パッケージに対する変更の種類 (続き)

パッケージに対する変更	変更が可能なパッケージの状態
外部スクリプトとプリスクリプトの追加、変更、削除 (モジュラーパッケージ)	パッケージは稼働していてもよい。 変更は、パッケージが稼働している場合もしていない場合も、適用されると有効になります。スクリプトを追加すると、Serviceguard はそのスクリプトを検証して、(エラーがなければ) ユーザーが変更を適用した時点で実行します。スクリプトを削除すると、Serviceguard はそのスクリプトを、ユーザーが変更を適用した時点で停止します。
パッケージの <code>auto_run</code> の変更	パッケージは、稼働していても停止していても構いません。 「切り替え動作とフェイルオーバー動作の選択」(96 ページ) を参照してください。
構成済みの依存関係の追加または削除	パッケージは稼働中または停止済みのどちらでもよい。 保守モードのパッケージには、特別な規則が適用されます。「保守モードまたは部分起動保守モードでのパッケージの依存関係規則」(207 ページ) を参照してください。 依存関係のため、再構成中のパッケージはアップとみなされます。つまり、pkgA が pkgB に依存し、また pkgA がダウンしていて pkgB が再構成されている場合に、的確な状態になれば、pkgB の再構成が完了していなくても、pkgA は起動します。 パッケージの依存関係の変更と、新しい依存元パッケージが依存するリソースやサービスに影響を与える変更は、別々に行うことをお勧めします。リソースおよびサービスを最初に再構成し、変更を適用してから、パッケージの依存関係を構成してください。 詳細は、「パッケージの依存関係について」(100 ページ) を参照してください。
<code>evaluation_type</code> が <code>during_package_start</code> の汎用リソースの追加	汎用リソースのステータスが「down」でなければ、パッケージは稼働していてもよい。汎用リソースのオンラインでの変更については、「汎用リソースのオンライン再構成」(100 ページ) を参照してください。
<code>evaluation_type</code> が <code>before_package_start</code> の汎用リソースの追加	汎用リソースのステータスが「up」の場合はパッケージを稼働していてもよいが、そうでない場合はパッケージを停止する必要がある。
汎用リソースの削除	パッケージは稼働していてもよい。
<code>generic_resource_evaluation_type</code> の変更	汎用リソースのステータスが「up」の場合はパッケージを稼働してもよい。 generic_resource_evaluation_type を変更するとパッケージが停止する場合は、許可されません。 汎用リソースのオンラインでの変更については、「汎用リソースのオンライン再構成」(100 ページ) を参照してください。
<code>generic_resource_up_criteria</code> の変更	新しい <code>up_criteria</code> がリソースステータスの評価が「down」に変わる原因にならないのであれば、<code>evaluation_type</code> が『before_package_start』または『during_package_start』のリソースに対してパッケージを稼働してもよい。 generic_resource_up_criteria を変更するとパッケージが停止する場合は、許可されません。 汎用リソースのオンラインでの変更については、「汎用リソースのオンライン再構成」(100 ページ) を参照してください。

警告が発せられる変更

以下を変更すると警告が発せられるので、変更によってパッケージが異常終了するような場合には、変更を取り消すことが可能です。

注記: `cmapplyconf -f` を使用している場合には、変更を取り消すことはできません。

- パッケージノード
- パッケージの依存関係

- パッケージの重み (およびクラスター構成ファイルで定義されたノードキャパシティ)
- パッケージの優先順位
- `auto_run`
- `failback_policy`

クラスターイベント発生時の対処

Serviceguard では、稼働中のシステム管理作業はあまり必要ではありません。障害がない限り、クラスターは常に監視され、保護されています。障害が発生しても、他のノードへ移動するよう指示しておいたパッケージは、自動的に移動されます。システム管理者として必要な稼働中の管理作業は、クラスターの監視と、パッケージの移動が発生したかどうかを確認することです。移動が発生した場合は、原因を究明して修正作業を行います。

パッケージが移動した場合の一般的な修正作業には、以下のものがあります。

- 移動発生時刻の確定。
- 移動の原因を特定する。
- ハードウェアの故障があれば修理する。
- すべてのソフトウェアに問題が発生した場合の修復。
- ノードを再起動する。
- 元のノードへのパッケージの移動。
- パッケージの切り替えの有効化。

単一ノードによる運用

マルチノードクラスターでは、1 つのノードを除きすべてのノードが故障し (または、1 つのノードを除きすべてのノードをシャットダウンし)、クラスターが単一ノードで運用される場合があります。残ったノードでは、おそらくアプリケーションが実行されています。Serviceguard のデーモン `cmclld` がアクティブである限り、他のノードをクラスターに再結合させることができます。

クラスターが単一ノードで運用中に Serviceguard デーモンが異常終了した場合は、ノードは稼働したまま、アプリケーションも実行されたままとなります。

注記: これは、Serviceguard 自体は動作していないことを意味します。

この状況では、アプリケーションはまだ動作しており、パッケージ切り替えに対応できるノードもないため、単一ノードを停止する必要はありません。(これはマルチノードクラスターで Serviceguard デーモンが停止した場合と異なります。マルチノードクラスターでは、ノードは停止し (システムリセット)、パッケージは引き継ぎノードに切り替えられます。)

Serviceguard の再起動はしないでください。単一ノードでまだ動作しているアプリケーションの新しいインスタンスが別のノードで起動されると、データが破損するおそれがあるためです。

クラスターを再起動する代わりに、適切な時期を選んでアプリケーションをシャットダウンし、その後ノードをリブートしてください。これにより、リブート後に Serviceguard によってクラスターが再起動されます。

システム上の Serviceguard の削除

Serviceguard を今後ノードで使わないようにするには、`rpm -e` コマンドを使ってソフトウェアを削除します。

- △ **注意:** 最初にノードをクラスターから削除します。まだクラスターのメンバーとなっているサーバーで `rpm -e` コマンドを実行すると、クラスターは停止し、クラスターが削除されます。

Serviceguard を削除するには、以下の手順を実行します。

1. ノードがクラスターのアクティブメンバーの場合には、まずノードを停止します。
2. ノードがクラスター構成に含まれている場合には、構成からノードを削除します。
3. 複数のノードで Serviceguard を削除する場合は、一度に 1 台のノードで `rpm -e` コマンドを実行します。

8 クラスターのトラブルシューティング

この章では、クラスター運用の確認、クラスターステータスの確認、ハードウェアの追加と交換方法、およびクラスターの問題解決について説明します。説明する項目は以下のとおりです。

- クラスターの運用テスト
- ハードウェアの監視 (236 ページ)
- ディスクの交換 (236 ページ)
- LAN カードの交換 (238 ページ)
- 障害が発生したクォーラムサーバーシステムの交換 (239 ページ)
- トラブルシューティングの手掛かり (240 ページ)
- 問題の解決 (243 ページ)
- Serviceguard Manager のトラブルシューティング (249 ページ)

クラスターの運用テスト

Serviceguard クラスターを構成したら、障害時にクラスターの各種構成要素が正しく動作することを確認しておくといでしょう。この項では、クラスターがパッケージ障害、ノード障害、または LAN 障害に適切に対応することを、次の手順に従ってテストします。

- △ **注意:** 以下のクラスターテスト手順では、クラスターの各構成要素に人為的に障害を発生させ、クラスターがこれらの障害に適切に対処できるかどうかを確認めます。したがって、テストの実施により、ノードやアプリケーションの可用性が一時的に中断される場合があります。

パッケージマネージャーのテスト

パッケージマネージャーが正しく動作しているかどうかをテストするには、クラスター内の各パッケージに対して次の操作を実行します。

1. 次のコマンドを入力して、パッケージのサービスの PID 番号を取得します。

```
ps -ef | grep <service_cmd>
```

`service_cmd` は、`service_cmd` パラメーター(171 ページ) を用いて、パッケージ構成ファイル(または従来の制御スクリプト)で指定された実行可能ファイルです。選択されたサービスには、`service_restart` のデフォルト値 (none) が必要です。

2. 次のコマンドを入力して、`service_cmd` の PID を抹消します。

```
kill <PID>
```

3. 次のコマンドを入力して、パッケージステータスを表示します。

```
cmviewcl -v
```

パッケージは、指定された引き継ぎノード上で実行されている必要があります。

4. パッケージを停止してから、`cmhaltpkg` コマンド、`cmmodpkg` コマンド、および `cmrunpkg` コマンドを使って、一次ノードに戻します。

```
cmhaltpkg <PackageName>
```

```
cmmodpkg -e <PrimaryNode> <PackageName>
```

```
cmrunpkg -v <PackageName>
```

実行しているデータベースに応じた適切な方法でデータベースを回復してください。

汎用リソースを使用してパッケージマネージャーをテストすることもできます。クラスター内の各パッケージに対して次の操作を実行します。

1. 次のコマンドを入力してパッケージで構成されている汎用リソースを取得します。

```
cmviewcl -v -p <pkg_name>
```

2. 次のコマンドを使用して汎用リソースのステータスを「DOWN」に設定します。

```
cmsetresource -r <res1> -s down
```

3. 次のコマンドを入力して、パッケージステータスを表示します。

```
cmviewcl -v
```

パッケージは、指定された引き継ぎノード上で実行されている必要があります。

4. パッケージをプライマリノードへ戻します (「[フェイルオーバーパッケージの移動](#)」(204 ページ) を参照してください)。

注記: この汎用リソース用に構成された監視スクリプトがあった場合は、その監視スクリプトも汎用リソースのステータスの設定を試みる可能性があります。

クラスターマネージャーのテスト

クラスターマネージャーが正しく動作しているかどうかをテストするには、クラスター内の各ノードに対して次の操作を実行します。

1. ノードの電源を切断します。
2. 構成されている他のノードで次のコマンドを入力して、クラスターが再編成される過程を確認します。

```
cmviewcl -v
```

ここでは、電源を切断されたノードが停止され、そのパッケージが他のノードに正しく切り替えられたことを確認します。

3. ノードの電源を投入します。
4. 次のコマンドを入力して、ノードがクラスターに再び結合していることを確認します。

```
cmviewcl -v
```

ノードはクラスターによって認識され、かつノード上でパッケージが**実行されてはいけません**。

5. パッケージを元のノードに戻します。

```
cmhaltpkg <pkgname>
```

```
cmmodpkg -e -n <originalnode>
```

```
cmrunpkg <pkgname>
```

実行しているデータベースに応じた適切な方法でデータベースを回復してください。

6. クラスター内の全ノードに対して、上記の手順を 1 ノードずつ行います。

ハードウェアの監視

高可用性システムの管理では、障害に対する監視を常に心がけるとよいでしょう。日常的に監視を続けることにより、障害の発生を可能な限り防ぎ、万一発生した場合にも迅速に対処することができます。ディスクの監視については、「[ディスク監視構成の作成](#)」(184 ページ)を参照してください。さらに、以下のハードウェアに関するエラーや警告を監視します。

- CPU
- メモリ
- NIC
- 電源
- すべてのケーブル
- ディスクインターフェイスカード

上記の中には、単に物理的にチェックするだけでよいものもありますが、より全体的に詳しく監視するためには、システムログファイル (/var/log/messages) に記録される構成済み全 HA デバイスのレポートを定期的にチェックします。デバイスに関するエラーがあれば、保守作業が必要です。

ディスクの交換

障害が発生したディスクの交換手順は、ディスク構成のタイプにより異なります。Smart Array 関連の問題については、使用している Smart Array のマニュアルを参照してください。

ディスクアレイ内の故障したディスクドライブの交換

故障したディスクドライブは、単純にアレイから取り除き、同じタイプの新しいドライブと取り替えることで交換できます。再同期は、アレイ自体が行います。再同期が完了するまでは、ディスクの性能に影響が出ることもあります。ディスクドライブのホットプラグ処理についての詳細は、ディスクアレイのドキュメントを参照してください。

ロック LUN の交換

使用できなくなったロック LUN を、クラスターの稼働中に交換することができます。これは、デバイスファイル名を変更しない場合は、クラスターを再構成することなく行うことができます。また、デバイスファイルの変更が必要な場合でも、クラスターの稼働中に再構成を行うことができます。

別のデバイスファイルを使う必要がある場合は、クラスター構成ファイル内のデバイスファイル名を変更しなければなりません。「[オンラインでのクラスターロック LUN 構成のアップデート](#)」(217 ページ) を参照してください。

- △ **注意:** 作業を開始する前に、すべてのノードで syslog に、次のようなメッセージが記録されていることを確認してください。

WARNING: Cluster lock LUN /dev/sda1 is corrupt: bad label. Until this situation is corrected, a single failure could cause all nodes in the cluster to crash.

すべてのノードでこのメッセージが記録されたら、次のコマンドのように、新しいクラスターロック LUN を指定します。

```
cmdisklock reset /dev/sda1
```

- △ **注意:** cmdisklock を使う前に、LVM から、デバイスが接続されているノード上のどのサブシステムからもデバイスが使われていないことを確認するのは、ユーザーの責任です。この注意を守らずに cmdisklock を使うと、データが失われる可能性があります。

注記: cmdisklock コマンドは、ロック LUN を修復したり交換するときのみ必要です。詳細は、cmdisklock (1m) のマンページを参照してください。

Serviceguard はロック LUN を 75 秒ごとにチェックします。cmdisklock コマンドを実行したら、アクティブなクラスターノードの syslog ファイルを確認してください (長くても 75 秒以内)。この時点で、ロックディスクが正常に戻ったことを示すメッセージが出力されています。

破局的な障害後の Persistent Reservation の取り消し

Persistent Reservation (PR) とその仕組みについての詳細は、「[Persistent Reservation について](#)」(65 ページ) を参照してください。

通常の状況では、パッケージが停止したとき、Serviceguard はすべての PR をクリアします。ただし、クラスターに破局的な障害が発生した場合には、復旧の一環としてユーザー自身がクリーンアップを行うことが必要な場合があります。\$SGCONF/scripts/sg/pr_cleanup スクリプトを使用してこれを行います (このスクリプトは \$SGCONF/bin/にもあります。Linux の各ディストリビューションにおける Serviceguard ディレクトリの位置については、「[Serviceguard のファイルの位置](#)」(122 ページ) を参照してください)。

LUN のデバイス特殊ファイル (DSF) か、DSF 名のリストを含むファイルを指定する次のようなスクリプトを起動します。

```
pr_cleanup lun -v -k <key> [-f <filename_path> | <list of DSFs>]
```

- **lun** を使用した場合、ボリュームグループではなく LUN が動作することを指定します。
- **-v** を使用した場合、スクリプトが実行するアクションとそのステータスを詳しく示した詳細出力を指定します。
- **-k <key>** を使用すると、クリア動作で使われるキーを指定します。
- **-f <filename_path>** を使用すると、<filename_path> が指定するファイルに、動作する DSF の名前が記載されることを指定します。各 DSF は、1 行に 1 つずつ記載する必要があります。
- <list of DSFs> は、-f <filename_path> を使用しない場合、1 つ以上の DSF をコマンド行に指定します。

例

次のコマンドは、ファイル/tmp/pr_device_list 内にリストされた一連の LUN 上のキー abc12 に登録されたすべての PR 予約をクリアします。

```
pr_cleanup -k abc12 lun -f /tmp/pr_device_list
```

pr_device_list には次のようなエントリーが含まれます。

```
/dev/sdb1  
/dev/sdb2
```

あるいは、次のようにコマンド行にデバイスファイル名を入力することもできます。

```
pr_cleanup -k abc12 lun /dev/sdb1 /dev/sdb2
```

次のコマンドは、ボリュームグループ vg01 のベース LUN 上の PR キー abcde に登録されたすべての PR 予約をクリアします。

```
pr_cleanup -k abcde vg01
```

注記: キーワード lun が含まれていないため、このデバイスはボリュームグループとみなされます。

LAN カードの交換

LAN カードを交換する必要がある場合は、以下の手順を実行します。このとき、クラスターを停止させる必要はありません。

1. cmhaltnode コマンドを使ってノードを停止させます。
2. 次のようにしてシステムをシャットダウンします。

```
shutdown -h
```

システムの電源を切ります。

3. 障害が発生した LAN カードを取り外します。
4. 新しい LAN カードを取り付けます。このとき、カードは同一タイプのものを使用し、取り外したカードと同じスロットに取り付けてください。
5. システムの電源を入れます。
6. システムが起動すると、Red Hat システムの場合は kudzu プログラムによってハードウェアの変更が検出され報告されます。変更を受け入れ、新しい LAN カードに必要な情報を追加します。SUSE システムでは、システムのブート後に YAST2 を実行し、新しい LAN カードの NIC の設定を調整します。古い LAN カードが「ボンド」のメンバーだった場合は、新しい LAN カードをボンドのメンバーにする必要があります。[「チャネルボンディングの構成 \(Red Hat の場合\)」 \(127 ページ\)](#) または [「チャネルボンディングの構成 \(SUSE の場合\)」 \(129 ページ\)](#) を参照してください。
7. 必要に応じて、cmrunnode コマンドを使用して、ノードをクラスターに復帰させます。ノードが自動的にクラスターに結合するように構成されている場合は、このステップを省略できます。

Serviceguard はクラスターのバイナリ形式構成ファイルに格納されている値からカードの MAC アドレス (LLA) が変わったことを検出し、クラスター内の他のノードに新しい MAC アドレスを通知します。この後、クラスターは正常に動作します。

クラスター構成を適用し直して、クラスターのバイナリ形式構成ファイルの MAC アドレスを更新することをお勧めします。オンラインでの再構成を行うには、以下の手順を実行します。

1. cmgetconf コマンドを次のように使用して、新しい ASCII 形式の構成ファイルを取得します。

```
cmgetconf config.conf
```

2. `cmapplyconf` コマンドを使用して、構成を適用し、新しいバイナリファイルをすべてのクラスターノードにコピーします。

```
cmapplyconf -C config.conf
```

この手順により、バイナリファイルが新しいMACアドレスでアップデートされ、`cmviewconf` コマンドおよび `ifconfig` コマンドの出力間のデータの不整合がなくなります。

障害が発生したクォーラムサーバーシステムの交換

クォーラムサーバーに障害が発生したり、そのクォーラムサービスを利用するクラスターでクォーラムサーバーを使用できなくなった場合でも、クラスターに障害が発生することはありません。しかし、クォーラムサーバーが使えなくなると、それ以外の障害が発生した場合のクラスターの脆弱性が増します。故障したクォーラムサーバーシステムを交換するには、以下の手順を実行します。この手順の実行にあたって、どのクラスターノードの構成も変更する必要はありません。

- ① **重要:** 交換作業の前に、参照している『HP Serviceguard Quorum Server Release Notes』が最新のバージョンであるか確認してください。これらは、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] から入手できます。また、同じ場所にある Quorum Server のホワイトペーパーも参照してください。

1. ネットワークから古いクォーラムサーバーを削除します。
2. 新しいシステムを準備し、故障したクォーラムサーバーと同じ IP アドレスとホスト名で構成します。
3. 新しいシステムにクォーラムサーバー ソフトウェアをインストールし、構成を行います。新しい QS 承認ファイル (`/usr/local/qs/conf/qs_authfile` など) には、必ず古いクォーラムサーバーに構成されていたすべてのノードを含めてください。QS 承認ファイルの構成に関する詳細は、`qs(1)` のマンページを参照してください。

注記: クォーラムサーバーは、起動時に承認ファイルを読み取ります。`qs_authfile` ファイルを変更した場合は、必ず以下のコマンドを実行してファイルの再読み取りを強制的に実行してください。たとえば、Red Hat ディストリビューションでのコマンドは次のとおりです。

```
/usr/local/qs/bin/qs -update
```

SUSE ディストリビューションでは、次のコマンドを実行します。

```
/opt/qs/bin/qs -update
```

4. 以下の手順でクォーラムサーバーを起動します。
 - `init q` を使ってクォーラムサーバーを実行します。
または
 - Quorum Server 用に別のクラスター内にパッケージを作成します (使用しているバージョンの Quorum Server のリリースノートを参照してください)。これらは、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] から入手できます。
5. 古いクォーラムサーバーを使っていたすべてのクラスター内のすべてのノードは、新しいクォーラムサーバーに接続するようになります。クォーラムサーバーを使っている任意のクラスターで `cmviewcl -v` コマンドを実行し、そのクラスター内のノードがクォーラムサーバーに接続されたことを確認します。

6. 新しいウォーラムサーバー上のウォーラムサーバー ログファイルには、そのウォーラムサーバーを使用する各クラスターに関する次のようなログメッセージが表示されます。
Request for lock /sg/<ClusterName> succeeded. New lock owners: N1, N2
7. ウォーラムサーバーが正しく構成されていることを確認するには、また、ノードとウォーラムサーバー間の接続を検証するには、クラスターノードから以下のコマンドを実行します。

```
cmquerycl -q <QSHostName> -n <Node1> -n <Node2> ...
```

指定したノードがウォーラムサーバーと通信できない場合は、コマンドはエラーメッセージを出力します。

△ 注意: 古いシステムを、古い IP アドレスのままネットワークに戻さないようにしてください。

注記: 古いウォーラムサーバーがダウンしていて、新しいウォーラムサーバーのセットアップを行っている最中は、以下のようになります。

- cmquerycl、cmcheckconf、および cmapplyconf コマンドは機能しません。
 - cmruncl、cmhaltcl、cmrunnode、および cmhaltnode コマンドは機能します。
 - メンバーがちょうど半分に分割されるような、ノードまたはネットワークの障害が発生した場合、ウォーラムサーバーがタイブレーカーとして機能しなくなるため、クラスター障害が発生します。
-

トラブルシューティングの手掛かり

次の項では、動作中のシステム状態を確認し、クラスターのステータスデータ、ログファイル、構成ファイルを調査することによりトラブルシューティングを行う方法について、以下の項目を説明します。

- パッケージ IP アドレスの確認
- システムログファイルの確認
- 構成ファイルの確認
- パッケージ制御スクリプトの確認
- cmquerycl と cmcheckconf の使用
- cmviewcl の使用
- LAN 構成の確認

パッケージ IP アドレスの確認

ifconfig コマンドを使用すると、LAN の構成を確認できます。ノード *ftsys10* の停止後に *ftsys9* でこのコマンドを実行すると、eth1 のハートビート IP アドレスとともに、パッケージ IP アドレスが eth1:1 および eth1:2 に割り当てられていることがわかります。

```
eth0      Link encap:Ethernet  HWaddr 00:01:02:77:82:75
          inet addr:15.13.169.106 Bcast:15.13.175.255 Mask:255.255.248.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:70826196 errors:0 dropped:0 overruns:1 frame:0
          TX packets:5741486 errors:1 dropped:0 overruns:1 carrier:896
          collisions:26706 txqueuelen:100
          Interrupt:9 Base address:0xdc00

eth1      Link encap:Ethernet  HWaddr 00:50:DA:64:8A:7C
          inet addr:192.168.1.106 Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2337841 errors:0 dropped:0 overruns:0 frame:0
```

```

TX packets:1171966 errors:0 dropped:0 overruns:0 carrier:0
collisions:6 txqueuelen:100
Interrupt:9 Base address:0xda00

eth1:1    Link encap:Ethernet  HWaddr 00:50:DA:64:8A:7C
          inet addr:192.168.1.200 Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:9 Base address:0xda00

eth1:2    Link encap:Ethernet  HWaddr 00:50:DA:64:8A:7C
          inet addr:192.168.1.201 Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:9 Base address:0xda00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Bcast:192.168.1.255 Mask:255.255.255.0
          UP LOOPBACK RUNNING MULTICAST  MTU:3924 Metric:1
          RX packets:2562940 errors:0 dropped:0 overruns:1 frame:0
          TX packets:2562940 errors:1 dropped:0 overruns:1 carrier:896
          collisions:0 txqueuelen:0

```

システムログファイルの確認

クラスターマネージャーとパッケージマネージャーからのメッセージは、システムログファイルに書き込まれます。ログファイルのデフォルトの位置は、Linux のディストリビューションによって異なります。Red Hat でのデフォルトは、`/var/log/messages` です。vi などのテキストエディターや `more` コマンドを使用すると、ログファイルに記録されているクラスターの履歴を確認することができます。

ログファイルには、以下の情報が記録されています。

- 実行されたコマンドとその出力
- クラスターの主要なイベント (エラーが発生した場合、または発生しない場合)
- クラスターのステータス情報

注記: Serviceguard の他にも、Linux 上で実行される多くの製品が、メッセージの保存にシステムログファイルを使用しています。システムログの使用の詳細については、Linux のドキュメントを参照してください。

システムログのエントリーの例

syslog ファイルから抜粋した以下のエントリー例は、pkg5_run スクリプトの問題により実行できなかったパッケージを示しています。詳細は `pkg5_run.log` に記録されています。

```

Dec 14 14:33:48 star04 cmcld[2048]: Starting cluster management protocols.
Dec 14 14:33:48 star04 cmcld[2048]: Attempting to form a new cluster
Dec 14 14:33:53 star04 cmcld[2048]: 3 nodes have formed a new cluster
Dec 14 14:33:53 star04 cmcld[2048]: The new active cluster membership is:
    star04(id=1) , star05(id=2) , star06(id=3)
Dec 14 17:33:53 star04 cmlvmd[2049]: Clvmd initialized successfully.
Dec 14 14:34:44 star04 CM-CMD[2054]: cmrunpkg -v pkg5
Dec 14 14:34:44 star04 cmcld[2048]: Request from node star04 to start
    package pkg5 on node star04.
Dec 14 14:34:44 star04 cmcld[2048]: Executing '/usr/local/cmcluster/conf/pkg5/pkg5_run
    start' for package pkg5.
Dec 14 14:34:45 star04 LVM[2066]: vgchange -a n /dev/vg02
Dec 14 14:34:45 star04 cmcld[2048]: Package pkg5 run script exited with
    NO_RESTART.
Dec 14 14:34:45 star04 cmcld[2048]: Examine the file
    /usr/local/cmcluster/pkg5/pkg5_run.log for more details.

```

次は、パッケージが正常に起動された例です。

```

Dec 14 14:39:27 star04 CM-CMD[2096]: cmruncl
Dec 14 14:39:27 star04 cmcld[2098]: Starting cluster management protocols.
Dec 14 14:39:27 star04 cmcld[2098]: Attempting to form a new cluster
Dec 14 14:39:27 star04 cmclconfd[2097]: Command execution message
Dec 14 14:39:33 star04 cmcld[2098]: 3 nodes have formed a new cluster
Dec 14 14:39:33 star04 cmcld[2098]: The new active cluster membership is:

```

```

star04(id=1), star05(id=2), star06(id=3)
Dec 14 17:39:33 star04 cmlvmd[2099]: Clvmd initialized successfully.
Dec 14 14:39:34 star04 cmcld[2098]: Executing '/usr/local/cmcluster/conf/pkg4/pkg4_run
start' for package pkg4.
Dec 14 14:39:34 star04 LVM[2107]: vgchange /dev/vg01
Dec 14 14:39:35 star04 CM-pkg4[2124]: cmmodnet -a -i 15.13.168.0 15.13.168.4
Dec 14 14:39:36 star04 CM-pkg4[2127]: cmrunserv Service4 /vg01/MyPing 127.0.0.1
>>/dev/null
Dec 14 14:39:36 star04 cmcld[2098]: Started package pkg4 on node star04.

```

構成ファイルの確認

以下の ASCII 形式の構成ファイルを確認します。

- クラスター構成ファイル
- パッケージ構成ファイル

構成を設計したときに使用したワークシートなどを参照して、ファイルの内容に記述もれや誤りがないことを確かめます。

パッケージ制御スクリプトの確認

従来のパッケージでは、パッケージを実行するノードには必ずパッケージ制御スクリプトが存在しますが、そのスクリプトファイルがすべてのノードで同一であることを確認してください。また、スクリプトがすべてのノードで実行可能であることを確認してください。さらに制御スクリプト名がパッケージ構成ファイルに存在すること、パッケージ構成ファイルで指定されたすべてのサービスがパッケージ制御スクリプトに存在することも確認してください。

各々のパッケージの起動と停止の情報は、パッケージの制御スクリプトログにあります。このログには、すべてのパッケージの起動動作や停止動作などの、パッケージ操作の履歴が記録されます。このファイルの位置は、パッケージ構成ファイルの `script_log_file` パラメーター(166 ページ)によって決まります。従来のパッケージで実行スクリプトと停止スクリプトを別々に記述した場合は、スクリプトごとにログが作成されます。

cmquerycl と cmcheckconf コマンドの使用

構成の確認に使用する `cmquerycl` と `cmcheckconf` は、クラスターのトラブルシューティングにも使用できます。次の例では、2 つのコマンドを使用して、`ftsys9` と `ftsys10` での既存のクラスターの構成を確認しています。

```

cmquerycl -v -C $SGCONF/verify.conf -n ftsys9 -n ftsys10
cmcheckconf -v -C $SGCONF/verify.conf

```

`cmcheckconf` では、以下の項目が確認されます。

- ネットワークのアドレスと接続
- クォーラムサーバーとの接続性 (クォーラムサーバーが構成されている場合)
- ロック LUN との接続性 (ロック LUN が使用されている場合)
- 次の内容に関するクラスターとパッケージの構成パラメーターの有効性
 - 名前の一意性
 - スクリプトの存在とパーミッション

なお、`cmcheckconf` コマンドでは以下の内容は確認されません。

- 電源システムの正しい設定
- パッケージ構成スクリプトの正当性

LAN 構成の確認

以下のネットワークコマンドを使用して、問題を診断できます。

- `ifconfig`: LAN 構成を調べます。このコマンドにより、各 LAN インターフェイスカードに割り当てられたすべての IP アドレスのリストが表示されます。
- `arp -a` は、arp テーブルをチェックするのに使用できます。
- `cmscancl`: クラスタ内のネットワークインターフェイス間の IP レベルの接続性をテストします。
- `cmviewcl -v`: 一次 LAN のステータスを表示します。

上記のコマンドを全ノードで使用してください。

問題の解決

Serviceguard で発生する問題は、次のカテゴリに分類されます。

- Serviceguard コマンドのハング
- クラスタの再編成
- システム管理エラー
- パッケージ制御スクリプトのハング
- パッケージの移動エラー
- ノード障害とネットワーク障害
- クォーラムサーバーのメッセージ

名前解決の問題

Serviceguard コマンドの多く (`cmviewcl` など) は、クラスタノードのアドレス検索をする際に名前解決サービスに依存します。名前解決サービスが使用可能でない場合 (ネームサーバーがダウンしている場合など) は、Serviceguard コマンドがハングしたり、ネットワークに関するエラーメッセージを返す場合があります。この状況が発生した場合は、各クラスタノード上で `host` コマンドを使って、名前解決が正しく実行されているか確認してください。例:

```
host ftsys9
```

```
ftsys9.cup.hp.com has address 15.13.172.229
```

上記のコマンドの出力にノードの正しい IP アドレスが含まれていない場合は、使用した名前解決サービスを調べてください。

ネットワークとセキュリティの構成エラー

多くの場合、`Permission denied...` や `Connection refused...` などの症状はネットワークとセキュリティの構成エラーによって起こります。このような問題のほとんどは、`/etc/hosts` のエントリーを修正することで解決できます。詳細は、「[名前解決の構成](#)」(124 ページ) を参照してください。

切り離されたパッケージの停止

`cmhaltpkg` を使って、切り離されたパッケージの停止を試みたときに、特定のノードに到達できないと、以下のエラーメッセージが表示されます。

```
Unable to halt the detached package <package_name> on node <node_name>
as the node is not reachable. Retry once the node is reachable.
```

このような場合、ノードは電源がオンで、アクセス可能のはずです。`cmhaltpkg` コマンドを再実行してください。

一時的な状況によって引き起こされるクラスターの再編成

次のような Serviceguard のエラーメッセージが表示される場合があります。これはノードに問題があることを示しています。

`Member node_name seems unhealthy, not receiving heartbeats from it.`

これは、ノード障害などの重大な問題を示している可能性があります。たとえば、`MEMBER_TIMEOUT` パラメーターの設定が小さすぎることが、根本的な原因であるかもしれません。次の項「[MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスターの再編成](#)」を参照してください。あるいは、ネットワーク通信量やシステム負荷が過剰であるなど、一時的な問題の場合もあります。

対処方法: 一時的なネットワークやシステムの負荷の問題のためにクラスターノードが異常停止している場合 (このため、ネットワークの中で、あるいは処理の間にハートビートメッセージの遅延が生じます)、可能な場合にはネットワークやシステム負荷の問題をまず解決する必要があります。それができなければ、次項で説明するように `MEMBER_TIMEOUT` の値を増やす方法があります。

MEMBER_TIMEOUT を極めて小さく設定することによって引き起こされるクラスターの再編成

`MEMBER_TIMEOUT` パラメーターを極めて低く設定している場合、クラスターデーモン `cmclld` は、問題を示す警告を `syslog` に書き込みます。特に注意すべき 3 つの警告を以下に示します。

1. `Warning: cmclld was unable to run for the last <n.n> seconds. Consult the Managing Serviceguard manual for guidance on setting MEMBER_TIMEOUT, and information on cmclld.`

つまり、`cmclld` が長時間、CPU を利用できなかったということです。これがクラスターの再編成中に生じると、1 つまたは複数のノードが異常終了する可能性があります。一部のコマンド (`cmhaltnode (1m)`、`cmrunnode (1m)`、`cmapplyconf (1m)` など) は、クラスターの再編成を伴うため、これらのコマンドの 1 つを実行するとノード障害を招く可能性があります。この可能性が高いほど、ハングが長くなります。

対処方法: このメッセージが月に 1 回以上現れる場合は、`MEMBER_TIMEOUT` を、報告された最大遅延の 10 倍以上に増やします。たとえば、報告されたメッセージで、`cmclld` が 1.6 秒の間に実行されなかったことがわかれば、`MEMBER_TIMEOUT` を 16 秒以上に増やします。

2. `This node is at risk of being evicted from the running cluster. Increase MEMBER_TIMEOUT.`

これは、ハングが長かったため、他のノードがハートビートの受信に遅延があることを認識し、ノードを「異常」と判断したことを示します。ここからクラスターからノードを除外する動作が開始されます。この動作の説明については、「[ノードがタイムアウトしたときの動作](#)」(68 ページ) を参照してください。

対処方法: また、これは、前項で説明したように一時的な問題を示す場合があります。このような問題を診断して修正し、再発しないことが確実であれば、これ以上の対策をする必要はありません。確実でない場合は、前項の手順と同様に `MEMBER_TIMEOUT` を増やします。

3. `Member node_name seems unhealthy, not receiving heartbeats from it.`
これは、前項と同様にノードが「異常」とであると判断されたことを示します。

対処方法: 第 2 項を参照してください。

要件や推奨事項などの詳細については、「[クラスター構成のパラメーター](#)」(82 ページ) に記載した `MEMBER_TIMEOUT` の説明を参照してください。

システム管理エラー

Serviceguard 構成時のエラーには、クラスターを起動するだけではわからないものが数多くあります。クラスターは正しく起動され、あらゆる機能が正常であるように見えますが、ハードウェアやソフトウェアに障害が発生したり、パッケージの制御が他ノードへ移行されないなどの障害が発生したときにはじめて、構成にエラーがあったことがわかります。

これらは、特に以下のようなクラスター構成ファイルやパッケージ構成スクリプトのエラーが原因で発生します。

- ボリュームグループが引き継ぎノードで定義されていない。
- 引き継ぎノードにマウントポイントがない。
- 引き継ぎノードでネットワークエラーが発生した (構成エラー)。
- 引き継ぎノードでユーザー情報が正しく定義されていない。

このような場合は、以下のコマンドを使用して、ディスクのステータスを調べることができます。

- `df` - パッケージのボリュームグループがマウントされているかどうか調べます。
- `vgdisplay -v` - すべてのボリュームが存在するかどうか調べます。
- `strings /etc/lvmconf/*.conf` - 構成が正しいことを確認します。
- `fdisk -v /dev/sdx` - ディスクに関する情報を表示します。

パッケージ制御スクリプトのハングまたは障害

`RUN_SCRIPT_TIMEOUT` または `HALT_SCRIPT_TIMEOUT` の値が設定されている場合に、制御スクリプトがハングしてタイムアウトを超えると、Serviceguard はスクリプトを強制終了して、パッケージを「Halted」とマークします。同様に、パッケージの制御スクリプトが異常終了すると、Serviceguard はスクリプトを強制終了して、パッケージを「Halted」とマークします。どちらの場合においても、以下の状態になります。

- パッケージの制御は移行されない。
- パッケージの実行または停止が最後まで正しく実行されない場合がある。
- グローバルスイッチが不能状態になる。
- 現在のノードでパッケージを実行できなくなります。

上記のような障害が発生すると制御スクリプトが終了され、特に以下のようなパッケージリソースが解放されないままになる場合があります。具体的には、次の点に注意してください。

- ボリュームグループ (非アクティブ化されない)。
- ファイルシステム (マウントが外されない)。
- IP アドレス (解除されない)。
- サービス (実行が停止されない)。

このような場合は、オペレーターによる介入操作がなければ Serviceguard はパッケージを再起動できません。以下の手順を参考にしてすべてのリソースをいったん解放した後、パッケージを再起動してください。

1. アプリケーションごとにクリーンアップを行います。制御スクリプトによって行われたアプリケーション固有のアクションをすべて取り消さなければ、パッケージを代替ノードで正しく起動することはできません。取り消しの作業には、アプリケーションプロセスのシャットダウン、ロックファイルの削除、一時ファイルの削除などが含まれます。
2. パッケージ IP アドレスをシステムから削除します。削除には `cmmodnet (1m)` コマンドを使用します。まず `ifconfig` コマンドを実行した結果の出力をチェックして、インストールされているパッケージ IP アドレスを調べます。パッケージ制御スクリプトに指定されて

いる IP アドレスのいずれかが `ifconfig` の出力の `ethX:Y` ブロックの `inet addr:` の欄にある場合は、`cmmodnet` コマンドを使ってその IP アドレスを削除します。

```
cmmodnet -r -i <ip-address>
          <subnet>
```

<ip-address> には上記の操作で表示されたアドレスを入力し、<subnet> には、`ifconfig` の出力の `inet` アドレスと同じ行にあるマスクで <ip-address> をマスクした結果を入力します。

3. パッケージのボリュームグループを非アクティブ化します。まずファイルシステムが使用しているパッケージ論理ボリュームがあれば、マウントを外します。そのために、`df -l` コマンドを実行して出力を調べます。パッケージ制御スクリプト内に `Lv[]` 配列変数で指定されているパッケージ論理ボリュームが "Filesystem" 欄にあれば、`umount` コマンドを使ってマウントを外します。

```
fuser -ku <logical-volume>

umount <logical-volume>
```

次に、パッケージのボリュームグループを非アクティブ化します。ボリュームグループは、パッケージ制御スクリプトに `Vg[]` 配列変数で指定されています。

```
vgchange -a n <volume-group>
```

4. 最後に、パッケージを切り替え可能状態にします。

```
cmmodpkg -e <package-name>
```

タイムアウトが発生したノード上でクリーンアップを行った後に、同じノードを代替ノードとしてパッケージを実行するのが望ましい場合は、そのノード上でパッケージをもう一度使用可能にして、パッケージを実行できるようにします。

```
cmmodpkg -e -n <node-name>
          <package-name>
```

Serviceguard のデフォルトの制御スクリプトは、アプリケーションの実行または終了に必要な処理だけを行うように作成されています。パッケージ管理者により処理実行時間の制限が設定されていて、何らかの理由により制限時間内に処理が終わらなければ、Serviceguard は通常制御スクリプトのロジックがハングしたか、ロジックに誤りがあると判断します。その時点で制御スクリプトの信頼性は失われ、クリーンアップ処理を正しく実行できないと判断され、スクリプトが終了されます。この場合パッケージ管理者は、どのようなクリーンアップが必要であるかを確認しなければなりません。

制御スクリプトのタイムアウトが発生した場合にパッケージを自動的に切り替えるには、`node_fail_fast_enabled` パラメータ(164 ページ) を `YES` に設定します。この場合 Serviceguard は、制御スクリプトのタイムアウトが起きたノードをリブートさせます。これにより、パッケージの実行または停止を試みた結果はすべて取り消され、設定に基づいて使用可能な代替ノード上でパッケージが自動的に再起動されます。

パッケージの移動エラー (従来のパッケージ)

パッケージ移動エラーは、特にパッケージ制御スクリプトのエラーが原因で発生すること以外、システム管理エラーに類似しています。エラーを防止する最善の方法は、高可用性アプリケーションを実行する前に、パッケージ制御スクリプトをテストすることです。

制御スクリプトの 2 行目に「set -x」を追加すると、スクリプトのエラー発生箇所の詳細が表示されます。

クリーンされない LVM2 ホストタグによるパッケージ起動の失敗

LVM2 ホストタグ機能をボリリュームグループで使用している場合、ホストタグはパッケージ停止プロセスのたびにクリーンアップされるように Serviceguard によって制御されています。ただし、ノードの電源障害、または SERVICE_FAIL_FAST/NODE_FAIL_FAST 機能によって発生したクラッシュの場合には、ホストタグがクリーンアップされません。このような場合、ホストタグは、他のノードでパッケージを起動する前に、手動でクリーンアップする必要があります。パッケージを他のノードで起動できなかった場合、パッケージログには次のようなメッセージが表示される場合があります。ホストタグをクリーンアップする手順も示されます。

```
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: ERROR: Function activation_check:
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: Error vg01 may still be activated on xyz.hp.com
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: To correct this situation, login to "xyz.hp.com " and
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: execute the following commands:
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: vgchange -a n vg01
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: vgchange --deltag xyz.hp.com vg01
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: Once "vg01" has been deactivated from "xyz.hp.com",
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: this package may be restarted via either cmmodpkg (1M)
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: or cmrunpkg(1M).
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: In the event that "xyz.hp.com" is either powered off
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: or unable to boot, then "vg01" must be forced
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: to be activated on this node.
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: ***** WARNING *****
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: Forcing activation can lead to data corruption if
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: "xyz.hp.com" is still running and has "vg01"
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: active. It is imperative to positively determine that
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: "xyz.hp.com" is not running prior to performing
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: this operation.
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: *****
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: To force activate "vg01", execute the following
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: command on the local system:
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: vgchange --deltag xyz.hp.com vg01
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: The package may then be restarted via either
Feb 11 17:18:36 root@abc.hp.com volume_group.sh[1871]: cmmodpkg (1M) or cmrunpkg (1M) commands.
```

ノード障害とネットワーク障害

ノード障害またはネットワーク障害が発生すると、Serviceguard は別のノードへパッケージの制御を移行させます。Serviceguard では制御の移行は正常な動作として処理されますが、管理作業上は移行の発生を認識し、クラスターを現状のままで維持するか、回復作業が必要であるかを判断する必要があります。

ノード障害は、以下のような原因により発生します。

- リブート
- カーネル障害
- ハング
- 電源異常

以下のコマンドを使用すると、ネットワークとサブネットのステータスを調べることができます。

- ifconfig - LAN のステータスを表示して、パッケージ IP が LAN カードにスタックされているかどうか調べます。
- arp -a - arp テーブルを調べます。

クラスターの構成にはさまざまな要素があり、それらはクラスターによって異なるので、すべての問題を一律に解決できる手順はありませんが、この章で説明したチェックやコマンドを実行し、ログファイルの内容を詳しく調べることにより、問題を識別して解決することができます。

クォーラムサーバーのトラブルシューティング

注記: クォーラムサーバーの構成についての詳細は、『HP Serviceguard Quorum Server Version A.04.00 Release Notes』を参照してください。必ずお使いのバージョンのリリースノートを読んでから先に進んでください。

承認ファイルの問題

Serviceguard ノードの syslog ファイルまたは `cmviewcl -v` の出力に次のようなメッセージが記録されているときには、アクセス承認の問題が発生している可能性があります。

```
Access denied to quorum server 192.6.7.4
```

承認ファイルをアップデートしていないことが原因と考えられます。ノードがファイルに含まれていることを確認し、`/usr/sbin/qs -update` を使ってクォーラムサーバーの承認ファイルをもう一度読み取らせてください。

タイムアウトの問題

Serviceguard ノードの syslog ファイルに次のようなメッセージが記録されているときには、タイムアウトの問題が発生している可能性があります。

```
Unable to set client version at quorum server 192.6.7.2: reply timed out
```

```
Probe of quorum server 192.6.7.2 timed out
```

このメッセージが表示される場合、ネットワークで間欠障害が発生しているか、またはデフォルトのクォーラムサーバータイムアウト値が小さすぎる可能性があります。

`QS_TIMEOUT_EXTENSION` を設定して、タイムアウト値を大きくするか、`MEMBER_TIMEOUT` 値を大きくしてください。これらのパラメーターについての詳細は、「[クラスター構成のパラメーター](#)」(82 ページ) を参照してください。

Serviceguard ノードの syslog ファイルに次のようなメッセージが記録されているときは、ノードがロック要求に対する応答を時間内に受信しなかったことを示しています。そのノードとクォーラムサーバー間の通信、またはクォーラムサーバーとクラスター内の他のノード間の通信に遅延が発生したことが原因と考えられます。

```
Attempt to get lock /sg/cluser1 unsuccessful. Reason: request_timedout
```

メッセージ

Serviceguard のコーディネータノードは、時々クォーラムサーバーに要求を送信し、ロック状態を設定します。(これは、タイブレークのロックを取得する要求とは異なります。) クォーラムサーバーの接続が確立されていないクラスターノードがある場合、この設定要求は失敗します。このとき、次のような 2 行のメッセージが、クォーラムサーバーのログファイルに記録されます。

```
Oct 008 16:10:05:0: There is no connection to the applicant
2 for lock /sg/lockTest1
Oct 08 16:10:05:0: Request for lock /sg/lockTest1 from
applicant 1 failed: not connected to all applicants.
```

この状態は、無視して構いません。この要求は数秒後に再試行され、成功します。次のメッセージがログに記録されます。

```
Oct 008 16:10:06:0: Request for lock /sg/lockTest1
succeeded. New lock owners: 1,2.
```

ロック LUN のメッセージ

ロック LUN デバイ스에 장애가 발생すると、syslog ファイルに次のメッセージが記録されます。

```
Oct 008 16:10:05:0: WARNING: Cluster lock lun /dev/sdc1 has failed.
```

Serviceguard Manager のトラブルシューティング

この項では、Serviceguard Manager に関連する問題についてトラブルシューティングを行う方法を説明します。

問題	解決策
Serviceguard Manager の起動を試みると「Service Temporarily Unavailable」と表示される	/etc/hosts ファイルに指定されているループバックアドレスが127.0.0.1 localhost.localdomain localhostであることを確認します。
Tomcat プロセスが開始されていないことがある	Tomcat 起動コマンド /opt/hp/hpsmh/tomcat/bin/startup.sh を実行します。
Java または Tomcat のバージョンが前提条件に従っていない場合、HP System Management Home (SMH) Web ページから Serviceguard Manager を起動すると以下のメッセージが表示される Service Temporarily Unavailable、Http status 500 proxy error、SMH can't find the requested page	1. 前提条件となっているバージョンの Java(1.6 以降) および Tomcat(5.x または 6.x) をインストールします。 2. /usr/bin/java が、サポートされるバージョンの Java ("ll /usr/bin/java") を指していることを確認します。指していない場合には、リンクを解除し (unlink /usr/bin/java)、新しいリンクを作成します (ln -s <full path of JAVA installation directory> /usr/bin/java)。 3. /usr/share/sgmgr-tomcat が、Tomcat のインストールディレクトリ、つまり catalina home ("ll /usr/share/sgmgr-tomcat") を指していることを確認します。指していない場合には、リンクを解除し (unlink /usr/share/sgmgr-tomcat)、新しいリンクを作成します (ln -s <full path of Tomcat installation directory> /usr/share/sgmgr-tomcat)。 4. Serviceguard-manager-tomcat rpm がインストールされていることを確認します (コマンド "rpm -qa grep serviceguard" を実行すると、"serviceguard-manager-tomcat-01.00-0" が表示されるはずです)。 5. /opt/hp/hpsmh/tomcat/bin/tomcat_cfg スクリプトを実行します。

A 高可用性クラスターアプリケーションの設計

この付録では、高可用性アプリケーションの作成/移植方法について、以下の項目に重点を置いて説明します。

- アプリケーション操作の自動化
- アプリケーションのフェイルオーバー速度の制御 (251 ページ)
- 複数のシステムで実行されるアプリケーションの設計 (254 ページ)
- クライアント接続の復元 (258 ページ)
- アプリケーション障害の処理 (259 ページ)
- 計画的ダウンタイムの短縮 (260 ページ)

高可用性向けの設計とは、ユーザーが経験する予期しないダウンタイムと計画的ダウンタイムを短縮することです。予期されないダウンタイムには、停電、システム障害、ネットワーク障害、ディスクのクラッシュ、またはアプリケーション障害などの予定されていないイベントがあります。計画的ダウンタイムには、バックアップスケジュールに基づくバックアップ、新しい OS リビジョンへのシステムアップグレード、またはハードウェア交換など予定されているイベントがあります。

以下に示す 2 つの主要な対策に留意してください。

1. システムのリブートやパニックに対応できるようにアプリケーションを設計してください。既存のアプリケーションを高可用性環境向けに変更する場合は、現時点で、システムパニックの発生後、アプリケーションがどのような状態になるかを確認します。高可用性環境では、アプリケーションを再起動する手順を定義(かつスクリプト化)しておくことが必要です。アプリケーションの起動/停止手順は、ユーザーによる介入が不要な、自動化された手順でなければなりません。
2. 以下に示すシステム固有の情報をアプリケーションで使用するにより、別のシステムへのフェイルオーバーや正常動作が妨げられる場合は、これらの情報を使用しないでください。
 - アプリケーションでは `uname()` または `gethostname()` を参照しないでください。
 - アプリケーションでは SPU ID を参照しないでください。
 - アプリケーションでは MAC (リンクレベル) アドレスを使用しないでください。

アプリケーション操作の自動化

アプリケーションは自動的に起動したり停止できるようになっていますか、あるいは、オペレーターの介入が必要でしょうか。

この項では、アプリケーション動作を自動化して、ユーザーの介入を不要にする方法を説明します。高可用性の第一の規則の 1 つは、手動による介入を回避することです。ユーザーが端末、コンソール、または GUI インターフェイスからコマンドを入力してサブシステムを起動する場合は、ユーザーはシステムの重要な要素になってしまいます。この場合、ユーザーがシステムコンソールがある場所へ来て、必要な作業を行うまでに何時間もかかる可能性があります。たとえば、問題が発生しているハードウェアが離れた場所で、しかも操作方法が分かるユーザーがいない場所にある場合や、厳重に管理されているデータセンターにシステムがある場合、または閑散時にモデムを経由して接続する必要がある場合などがあります。

アプリケーションの再配置を自動化する際は、以下の 2 つの原則を考慮してください。

- ユーザーが故障停止の影響を受けないようにすること。
- アプリケーションに起動/停止手順が定義されていること。

アプリケーションが実行されているシステムがリブートされた場合どのような状態になるか、また高可用性に対するアプリケーションの動作を変更する必要があるかどうかを認識しておくことが必要です。

ユーザーが故障停止の影響を受けないようにする

ユーザーが故障停止の影響をできるだけ受けないようにしてください。このためには、以下の項目を考慮します。

- サーバー障害が原因で接続が切断された場合、再接続するためにユーザーの介入を必要としないようにします。
- 実行中のフェイルオーバーによるわずかな遅延も、できる限りユーザーに警告します。
- データの再入力を最小限に抑えます。
- 予備容量を考慮したシステム設計を行い、ユーザーに与える性能低下を最小限に抑えます。

アプリケーションの起動/シャットダウン手順の定義

アプリケーションは、手動による介入なしで再起動できることが必要です。アプリケーションを起動する際にハードウェアのスイッチを入れることが必要な場合は、再起動の自動化は不可能です。アプリケーションの起動、シャットダウン、監視の手順は、HA ソフトウェアが自動的に実行できるように作成することが必要です。

アプリケーションの応答を確実に自動化するには、アプリケーションの起動/停止手順が定義されていることが必要です。Serviceguard では、起動/停止手順はパッケージ制御スクリプトに記述されています。これらの手順は、エラーの有無をチェックして、HA 制御ソフトウェアにステータスを返すことが必要です。すべての応答を前もって決定したりスクリプト化できない場合は、起動と停止をコマンド行から実行する手順とし、対話形式で行わないようにしなければなりません。

HA フェイルオーバー環境では、HA ソフトウェアは、必要なリソース (必要なディスクドライブへのアクセス機能など) を持つ、クラスター内のアクティブなシステムでアプリケーションを再起動します。以下の 2 つの方法でアプリケーションを再起動することが必要です。

- アプリケーションは、バックアップシステム上 (またはアプリケーションの再起動オプションが選択されている場合は同じシステム上) で再起動と回復できなければなりません。
- アプリケーションが起動中に異常終了し、障害の原因が解決された場合は、アプリケーションは再起動できなければなりません。

アプリケーションの管理者は、適切な HA コマンドを使用して、アプリケーションの起動/シャットダウン手順を習得することが必要です。これは、誤ってアプリケーションを直接シャットダウンすると、不要なフェイルオーバーが発生するためです。また、アプリケーション管理者は、開発環境のテスト用インスタンスと間違えて、本番用インスタンスを誤ってシャットダウンしないように注意しなければなりません。

HA ソフトウェアがアプリケーションで障害が発生したことを認識できるように、アプリケーションがアクティブかどうかを監視する機構が必要です。これは、アプリケーションに属するすべてのプロセスに対して `ps -ef | grep xxx` コマンドを実行するスクリプトと同じくらい簡単なスクリプトで対応できます。

ユーザーへの影響を少なくするために、アプリケーションはエラーが発生した場合でも簡単に処理を中断してはいけません。これは、このような中断によりバックアップシステムへの不要なフェイルオーバーが発生するためです。したがって、アプリケーションは、エラーの発生時にすぐに処理を中断しないで、エラーの内容を正確に判断して適切なエラー回復処置を実行することが必要です。

アプリケーションのフェイルオーバー速度の制御

フェイルオーバーを最短時間で実行するにはどのような手順が必要でしょうか。

障害が発生してアプリケーションが別のノードへ移動 (フェイルオーバー) される場合、アプリケーション側でさまざまな項目を設計しておくことにより、アプリケーションをバックアップして実行するのに要する時間を短縮することができます。ここでは以下の項目について説明します。

- 非データファイルシステムの複写
- raw ボリュームの使用

- ジャーナルファイルシステムの使用の評価
- データ損失の最小化
- 再開可能なトランザクションの使用
- チェックポイントの使用
- 複数サーバーの設計
- 複製データサイトの設計

非データファイルシステムの複写

非データファイルシステムは、共有するより複写して使用してください。アプリケーションデータのコピーは1つしか存在できません。このデータは、アプリケーションを実行するシステムがアクセスするディスクセット上に置かれます。これらのデータディスクがファイルシステムである場合は、フェイルオーバー後にファイルシステム回復 (fsck) を完了しなければ、データへはアクセスできません。ファイルシステムが小さいほど回復は速くなるため、回復時間の短縮に役立ちます。したがって、複写可能なものはデータファイルシステムに置かないことが最良の方法です。たとえば、アプリケーションの実行可能ファイルのコピーは、共有ファイルシステム上に置かないで、各システムに置いておく方が良いでしょう。また、アプリケーション実行ファイルを複写しておく、必要なときにアプリケーションの段階的アップグレードがしやすくなります。

ジャーナルファイルシステム (JFS) 使用の評価

ファイルシステムを使用する必要がある場合は、JFS の方が HFS に比べてはるかに速くファイルシステムを回復できます。ただし、JFS の性能はアプリケーションによって異なります。適切な JFS の例としては、Reiser FS (reiserfs は Serviceguard A.11.20.00 ではサポートされません)、ext3、ext4 などがあります。

データ損失の最小化

予期されない故障停止の際に失われる可能性のあるデータの量を最小限に抑えてください。障害が発生した場合、一部のデータが失われることは避けられません。ただし、次項で説明するとおり、失われるデータの量を最小限に抑える処置を取ることをお勧めします。

メモリベースのデータの使用とデータの量を最小限にする

障害が発生すると、メモリ上のデータ(メモリ上のコンテキスト)はすべて失われます。メモリ上のデータが容易に再計算できない場合は、メモリ上のデータの量を最小限に抑えるようアプリケーションを設計することが必要です。アプリケーションがスタンバイノード上で再起動した場合、メモリ上に必要な情報をすべて再計算するかディスクから再度読み込まなければなりません。

フェイルオーバーの速度を測定する方法の1つは、リブート後に通常のシステム上でアプリケーションが起動に要する時間を計算することです。アプリケーションはすぐに起動しますか。またはエンドユーザーがアプリケーションに接続できるようになるまでに、アプリケーションが処理しなければならない手順が多数ありますか。アプリケーションは、メモリ上のデータ構造体やテーブルを再度初期化する必要なく迅速に起動することが理想です。

性能の点から言えば、データはディスクに書き込むより、メモリに保存する方が良いでしょう。ただし、データの損失に伴う危険性と、データをディスクに保存することによる性能への影響とを比較してください。

共有ディスクからメモリに読み込んだ後、読取り専用として使用する場合、心配は不要です。

ログのサイズを小さくする

データベースによっては、ログをメモリバッファに書き込んでオンライン性能を向上させることができます。もちろん障害発生時には、処理中のトランザクションはすべて失われます。ただし、このメモリ上のログのサイズを最小化しておけば、障害発生時に失われる、終了したトランザクションのデータの量を少なくすることができます。

ディスク上のログのサイズを小さくしておけば、ログのアーカイブまたは複写をより頻繁に作成できるため、障害が発生した場合のデータ損失の危険性を軽減できます。もちろん、オンライン性能の向上とログのサイズの最小化との間にはトレードオフがあります。

ローカルデータの必要性の排除

ローカルデータの必要性は可能な限り排除してください。三層構造のクライアントサーバー環境では、中央の層にデータがない(つまり、クライアント固有のローカルデータ、または変更を必要とするローカルデータがない)場合がしばしばあります。このような場合、この「アプリケーションサーバー」の層には、より高いレベルの可用性、負荷のバランス設定、フェイルオーバーが実現します。ただし、このためにはすべてのデータをクライアント(第1層)またはデータベースサーバー(第3層)に保存することが必要です。

再開可能なトランザクションの使用

サーバーで障害が発生して、アプリケーションが別のシステムで再起動される場合、クライアントがトランザクションを再度入力したり、トランザクションを取り消す必要がないように、トランザクションは再開可能であることが必要です。つまり、トランザクションの途中で障害が発生した場合、最初からもう一度やり直す必要があってははいけません。この機能により、アプリケーションが強化され、ユーザーがフェイルオーバーを認識する回数が減ります。

一般的な例は、印刷ジョブです。プリンターアプリケーションは通常、ジョブのスケジュールを設定します。ジョブが終了すると、スケジューラは次のジョブを実行します。ただし、長いジョブ(給料明細の印刷に3時間かかる例など)の実行中にシステムがダウンした場合、システムが再起動したときどのような状態になるでしょうか。ジョブは最初から再開されもう一度すべての給料明細を印刷するのでしょうか、それとも残りの部分から開始するのでしょうか、またはスケジューラはジョブが完了したとみなし、残りは印刷しないのでしょうか。高可用性環境で正しく動作していれば、残りの部分からジョブが再開され、給料明細は1人に1枚ずつしか印刷されません。

もう1つの例として、事務員が新しい従業員に関するデータを入力するアプリケーションを挙げます。このアプリケーションでは各従業員に固有の番号が必要で、新しい従業員の名前と番号を入力した後障害が発生したと想定します。従業員番号は障害発生前に入力されていたので、番号の再入力をアプリケーションは拒否するのでしょうか。また、部分的に入力した情報を最初に削除するよう要求するのでしょうか。高可用性環境では、事務員はもう一度入力をやり直すか、次のデータ項目から続けて入力するか、必要に応じて適切な方法を取ることができません。

チェックポイントの使用

複雑なトランザクションについては、チェックポイントを取るようにアプリケーションを設計してください。ユーザー側から見ると1つのトランザクションでも、実際は複数のデータベーストランザクションである場合があります。この問題は、再開可能なトランザクションに関係していますが、システム障害によって中断されたトランザクションをフェイルオーバー発生後に完了できるように、ここではトランザクションの進行状況をローカル側で記録することをお勧めします。

たとえば、使用しているアプリケーションが n (円周率) を計算していると仮定します。元のシステムで、アプリケーションは小数点以下1,000桁まで計算を完了していますが、アプリケーションがディスクにまだ何も書き込んでいないとします。ちょうどその時に、ノードがクラッシュしたとします。アプリケーションは2番目のノードで再起動されますが、アプリケーションは一番最初から開始されるため、1,000桁分を再計算しなければなりません。アプリケーションがディスクに定期的に計算結果を書き込んでいれば、アプリケーションは中断された箇所から再開できたはずですが。

チェックポイント回数と性能のバランス

チェックポイント回数と性能とのバランスを取ることは重要です。ディスクへのチェックポイント回数を増やすと、性能に与える影響が大きくなります。チェックポイントを頻繁に行うとアプリケーションの処理速度が低下するのは明らかですが、チェックポイントの回数が不十分な場合、フェイルオーバー後にアプリケーションを現在の状態に戻すのに時間がかかります。したがって、エンドユーザーがチェックポイント回数を決めるのが理想的です。このために

は、エンドユーザーがチェックポイントの回数を調整できるようカスタマイズ可能なパラメータを用意することが必要です。

複数サーバーの設計

アクティブなサーバーを複数個同時に使用する場合、複数のサービスポイントにより比較的透過的なサービスがクライアントへ供給されます。ただしこの機能を使用するには、クライアントが、複数のサーバーとそのアドレスを指定する際の優先順位について十分な知識を持っていることが必要です。また、障害が発生したサーバーのデータまたは複製されたデータがアクセス可能なことも必要です。

たとえば、1 番目のシステムがダウンした場合にアプリケーションを 2 番目のシステムへ移行するより、両方のシステムでそのアプリケーションを実行するようにしてください。1 番目のシステムで障害が発生した場合、2 番目のシステムは単に 1 番目のシステムの業務を引き継げばよいのです。こうすれば、アプリケーションを起動する時間を省略できます。このような構造を設計する方法は多数ありますが、この種の設計に関する問題点も多数あります。ここでは、いくつかの例を挙げるだけに留め、詳しい説明は行いません。

最も簡単な方法は、2 つのアプリケーションをマスターとスレーブの関係で実行することです。この場合、スレーブは単にマスターに対してホットスタンバイの状態にあるアプリケーションとなります。マスターで障害が発生した場合、2 番目のシステムであるスレーブはデータがどのような状態にあったかを調べる必要があります (つまり、データ回復が必要です)。その場合でも、アプリケーションを fork して初期起動する時間は節約されます。

もう 1 つの方法は、2 つのアプリケーションをともにアクティブにする方法です。たとえば、データベースを提供する 2 つのアプリケーションサーバーがあるとします。クライアントの半数が一方のアプリケーションサーバーに接続して、残り半数はもう一方のアプリケーションサーバーに接続します。一方のサーバーがダウンした場合は、すべてのクライアントがもう一方のアプリケーションサーバーに接続すればよいのです。

複製データサイトの設計

複製データサイトは、迅速なフェイルオーバーと障害からの回復のどちらにも役立ちます。複製データを使用すれば、データディスクはシステム間で**共有されない**ため、データ回復を行う必要がありません。したがって回復時間が速くなります。ただし、データを複製することで性能が低下するというトレードオフがあります。データの複製方法は多数ありますが、アプリケーションの設計者は十分に検討して決定する必要があります。

標準データベース製品の多くは、データ複製がクライアントアプリケーションに対して透過的になっています。標準データベースを使用するようにアプリケーションを設計することにより、エンドユーザーはデータ複製が必要かどうかを決めることができます。

複数のシステムで実行されるアプリケーションの設計

アプリケーションがダウンしてバックアップノードへ移行した場合、アプリケーションはそれまでとは別のシステムでどのように動作するのでしょうか。

前項では、アプリケーションを確実に自動的に再起動するための方法を説明しました。この項では、アプリケーションを複数のシステムで実行するいくつかの方法を説明します。説明する項目は以下のとおりです。

- ノード固有情報の回避
- アプリケーションへの固有の名前の割り当て
- `uname (2)` の慎重な使用
- 固定ポートへのバインド
- 再配置可能 IP アドレスへのバインド
- 各アプリケーションへの独自のボリュームグループの割り当て
- SNA アプリケーションへの複数のあて先の使用
- ファイルロックの回避

ノード固有情報の回避

通常、新しいシステムをインストールするとき、アクティブな各ネットワークインターフェイスに IP アドレスを割り当てる必要があります。この IP アドレスは、常に特定のノードに関連付けられ、定常 IP アドレスと呼ばれます。

高可用性アプリケーションを持つパッケージを使用する場合は、IP アドレスのセットの追加が必要になります。このアドレスセットはアプリケーション自体に割り当てるためのものです。これは、アプリケーションの再配置可能 IP アドレスと呼ばれます。Serviceguard のネットワークセンサーは、この再配置可能なアプリケーション IP アドレスがあるサブネットへのノードのアクセスを監視します。Serviceguard でパッケージが構成されている場合、与えられたサブネットワークアドレスがパッケージ依存性項目として指定され、パッケージを実行できるノードのリストも提供されます。パッケージをリモートノードへフェイルオーバーする場合、サブネットワークはこのターゲットノード上ですでにアクティブになっていることが必要です。

各アプリケーションまたはパッケージには、再配置可能 IP アドレスだけでなく、固有の名前を付けることが必要です。この規則に従うことにより、アプリケーションが実行されるシステムと、アプリケーション自体とを分離することができるため、ユーザーはアプリケーションがどのシステムで実行されているかを知る必要がなくなります。また、負荷のバランス設定またはその他の目的により、クラスター内の異なるシステム間でアプリケーションを移動しやすくなります。2 つのアプリケーションが 1 つの IP アドレスを共有している場合は、これらのアプリケーションを同時に移動する必要があります。ただし、個別の名前とアドレスを指定しておけば、アプリケーションを別々に移動できます。

外部からクラスターへアクセスする場合、クライアントはアプリケーションの参照方法を知っておく必要があります。1 つの方法は、目的のアプリケーションに関連する再配置可能 IP アドレスをクライアントに通知することです。もう 1 つの方法は、アプリケーションをホストとみなして、ドメインネームシステム (DNS) でホスト名対アドレスのマッピングを構成することです。どちらの場合も、クライアントは結局、アプリケーションの再配置可能 IP アドレスを使って通信することになります。アプリケーションが別のノードへ移動すると IP アドレスも移動するので、クライアントはアプリケーションの現在の位置を知らなくても、アプリケーションを使用できます。各ネットワークインターフェイスは、定常 IP アドレスが与えられていることが必要です。この IP アドレスは、ネットワーク障害が発生してもリモートシステムへ**移動しません**。

十分な IP アドレスの獲得

各アプリケーションは、システム自体に割り当てられている定常 IP アドレスとは別に**再配置可能な** IP アドレスを受け取ります。したがって、1 つのシステムは、複数の IP アドレス、つまりシステム自体の IP アドレスとシステムが通常実行する各アプリケーションの IP アドレスを持つことになります。このため、サブネット内の IP アドレスは、高可用性を持たない場合と比べてたくさん消費されます。IP アドレスを余分に確保しておく必要があります。

同じネットワークインターフェイスに複数の IP アドレスを持つことができるのは、これらのアドレスが同じサブネットワーク上にある場合に限りです。

同一システム上の複数インスタンスの生成

それぞれ独自のアプリケーション名と IP アドレスを持つ複数のインスタンスを単一のシステム上で実行できるように、アプリケーションを作成することが必要です。この場合、アプリケーションを起動する際に、どのインスタンスが実行中であるかを示すパラメーターを使用することが必要になります。これにより、通常の状態では複数のシステム間にユーザーを分散させることができる一方、システムで障害が発生した場合はすべてのユーザーを保守の対象とすることができます。

SPU ID または MAC アドレスの使用回避

アプリケーションは、SPU ID または MAC (リンクレベル) アドレスを使用しないように設計してください。SPU ID は、不揮発性メモリ内にある固有のハードウェア ID で、変更不可能です。MAC アドレス (NIC ID とも呼ばれます) は、LAN ハードウェアに関連するリンク専用アドレスです。これらのアドレスの使用に関する問題は、ライセンスサーバーで頻繁に発生します。これは、ライセンスサーバーが、安全上の理由により、ライセンスが複数のノードにコピーされないようにハードウェア専用 ID を使用するからです。1 つの解決策として、複数のライセン

スを持つ方法があります。つまり、アプリケーションを実行する各ノードごとに1つのライセンスを与えます。もう1つの方法は、クラスター全体のSPU IDまたはノード名のセットの一覧を表示する機構を持つ方法です。指定されたセットに含まれているシステムでアプリケーションが実行されている場合、ライセンスは承認されます。

従来のHAソフトウェアには、サービスがバックアップシステムに移動した場合、IPアドレスとともにネットワークカードのMACアドレスも移動するものがあります。このような動作をServiceguardで行うことはできません。

アプリケーションでMACアドレスを使用した過去事例のうち、主な2つの理由を以下に示します。

- 発信元とあて先の間にあるネットワーク装置(ルーターなど)が旧世代に属するもので、MACアドレスとIPアドレスのペアを使って手動でプログラミングすることが必要だったため。この問題は、フェイルオーバーの際にMACアドレスをIPアドレスとともに移動することにより、解決できます。
- システムのダウンによるタイムアウトが原因でネットワーク装置のキャッシュが更新される間、最高20分の遅れが生じるおそれがあったため。この問題は、現在のHAソフトウェアでは、新しいMACアドレスを使って古いIPアドレスを新しくARP変換したものをブロードキャストすることによって解決できます。

アプリケーションへの固有の名前の割り当て

各アプリケーションには固有の名前を割り当てる必要があります。次にその名前を `gethostbyname(3)` への入力として使用できるようにDNSで構成する必要があります。以下の項を参照してください。

DNSの使用

DNSは、ホスト名をIPアドレスに割り当てたり、その逆を行う際に使用するAPIを提供します。これは、ターゲットシステムの名前を最初に指定するtelnetなどのBSDソケットアプリケーションに便利です。次にアプリケーションは、この名前をIPアドレスに割り当てて接続を確立しなければなりません。ただし、呼び出しの中には、使用時に注意が必要なものがあります。

アプリケーションは公式のホスト名またはIPアドレスを**参照してはいけません**。公式のホスト名とそれに対応するIPアドレスとは、プライマリLANカードとそのカードの定常IPアドレスを指します。したがって、ホスト名またはプライマリIPアドレスを参照または要求するどのアプリケーションも、HA環境では動作しない場合があります。この環境では、アプリケーションをサポートするシステムのネットワークIDは、あるシステムから別のシステムに移動しますが、ホスト名は移動しないためです。

これに関する問題を見つける方法の1つは、アプリケーションで `gethostname(2)` の呼び出しを見つけることです。HAサービスで `gethostname()` を使用する場合は注意が必要です。これは、アプリケーションが移動する場合、時間の経過によってアプリケーションの応答が変化する可能性があるからです。また、`gethostname()` を使用して `gethostbyname(3)` への呼び出し名を決定するアプリケーションも、同様の理由で使用を避けてください。また、定常IPアドレスを使って呼び出す場合も、`gethostbyaddr()` 呼び出しの応答は経過時間によって異なる場合があります。

その代わり、アプリケーションは、ホスト名と定常IPアドレスではなくアプリケーション名と再配置可能IPアドレスを常に参照する必要があります。アプリケーションが `gethostbyname(3)` を呼び出す場合は、ホスト名ではなくアプリケーション名を指定する方が適切です。これにより `gethostbyname(3)` は、アプリケーションのIPアドレスを渡します。このIPアドレスは、アプリケーションとともに新しいノードへ移動します。

ただし、アプリケーション名がDNSで構成されている場合に限り、アプリケーションのIPアドレスをつきとめる際に `gethostbyname(3)` を使用する必要があります。個々のHAサービスには、個別のアプリケーション名を付けるのが最良の方法と思われます。そうすれば、他のアプリケーションに影響を与えずに、各アプリケーションとそのIPアドレスを別のノードへ移動できます。DNSのホスト名には、定常IPアドレスだけを関連付ける必要があります。

uname(2) の慎重な使用

前項で説明したホスト名に関連する問題として、アプリケーションによる `uname(2)` の使用があります。`uname(2)` は公式のシステム名を返します。システム名は、システム内の LAN カードの数に関係なく、システムに固有の名前です。慣例では、`uname` と `hostname` は同じですが、必ずしも同じである必要はありません。アプリケーションによっては、システムへ接続した後、安全上の理由により `uname(2)` を呼び出して、それらが実際に正しいシステム上にあることを確認します。これは HA 環境では適切な方法とはいえません。というのは、サービスはあるシステムから別のシステムに移動しますが、`uname` と `hostname` はどちらも移動しないためです。したがって、アプリケーションでは、実行されている場所の確認をする別の手段を開発する必要があります。たとえば、構成ファイルに記述されているホスト名のリストをアプリケーションがチェックする方法です。

固定ポートへのバインド

ソケットをバインドする場合、固定したポートアドレスを指定するか、またはポートアドレスを動的に割り当てることができます。任意のポートにバインドすると、アプリケーションを後で別のクラスターノードで再起動したときに別のポートが割り当てられる可能性があります。これは、このアプリケーションにアクセスするクライアントを混乱させるおそれがあります。

このような問題を避けるため、ポート番号を動的に割り当てる代わりに、アプリケーションが実行されるすべてのノード上で同一の固定ポートを使用することをお勧めします。これにより、アプリケーションは、現在どのノードがアプリケーションを実行しているかに関係なく、常に同じポート番号を返します。アプリケーションのポート割り当ては `/etc/services` に保存して、割り当ての経過を追跡し、他のユーザーが同じポート番号を選択しないようにしなければなりません。

再配置可能 IP アドレスへのバインド

ソケットがバインドされる際に、ポート番号とともに IP アドレスが指定されます。これにより、通信に使用する IP アドレスを示し、アプリケーションがクライアントと通信できるインターフェイスを制限できるようにします。アプリケーションは、メッセージがどのインターフェイスにも到達できることを示す `INADDR_ANY` へバインドできます。

ネットワークアプリケーションは、定常 IP アドレス、再配置可能 IP アドレス、または `INADDR_ANY` にバインドできます。定常 IP アドレスが指定されている場合、アプリケーションを別のノードで再起動すると異常終了する場合があります。これは、定常 IP アドレスが新しいシステムに移動していないためです。アプリケーションが再配置可能 IP アドレスにバインドしている場合は、アプリケーションを別のシステムに移動した場合も正常に動作します。

サーバー型のアプリケーションの多くは `INADDR_ANY` にバインドします。これはアプリケーションがどのインターフェイス上でも要求を受信できることを意味します。これにより、クライアントは定常 IP アドレスまたは再配置可能 IP アドレスを送信できます。ただしこの場合、ネットワークコードはどの送信元 IP アドレスが応答に最も適しているかを決定できないため、必ず定常 IP アドレスを選択します。

TCP ストリームソケットの場合、TCP レベルのプロトコルスタックはコネクション型プロトコルのため、クライアントに代わってこの問題を解決します。クライアント側では、TCP は定常 IP アドレスを無視し、クライアントが元々使用していたバインド済みの再配置可能 IP アドレスを引き続き使用します。

ただし、UDP データグラムソケットを使用した場合は問題があります。クライアントは、再配置可能 IP アドレスを使って複数のサーバーに接続したり、サーバーの応答メッセージに含まれる発信元 IP アドレスに基づいて応答を選別します。ただし、応答に含まれる発信元 IP アドレスは定常 IP アドレスで、アプリケーションの再配置可能 IP アドレスではありません。したがって、UDP ソケットを作成してリスニングを行う場合、アプリケーションは、`INADDR_ANY` ではなく適切な再配置可能なアプリケーション IP アドレスを使って `bind(2)` を呼び出さなければなりません。

connect() の前に bind() を呼び出す

アプリケーションが接続を開始するとき、まず、アプリケーション IP アドレスを指定して `bind(2)` を呼び出した後、`connect(2)` を呼び出します。そうしなければ、接続要求は、希望する再配置可能なアプリケーション IP アドレスではなく、システムの送信 LAN インターフェ

イスの定常 IP アドレスを使って送信されます。この場合クライアントは、accept (2) 呼び出しからこの IP アドレスを受け取るため、クライアントソフトウェアに混乱を招き、正しく動作しなくなります。

各アプリケーションへの独自のボリュームグループの割り当て

データを使用する各アプリケーションには個別のボリュームグループを使用してください。アプリケーションがディスクを使用しない場合は、個別のボリュームグループを割り当てる必要はありません。ボリュームグループ(ディスクのグループ)は、ノード間で移動可能なストレージのユニットです。各アプリケーションがそれぞれのボリュームグループに割り当てられている(つまり、2つのアプリケーションが同じディスクドライブのセットを共有していない)場合、負荷のバランスを最も柔軟に設定できます。2つのアプリケーションが同じボリュームグループを使ってデータを保存する場合は、これらのアプリケーションを同時に移動しなければなりません。アプリケーションが蓄積したデータが別々のボリュームグループにある場合は、フェイルオーバー時に別のノードへの切り替えが行われます。

アプリケーションデータは、複数のディスクドライブと(可能な場合は)複数のマウントポイントに設定することが必要です。また、複数のディスクと個別のマウントポイントを使用できるようにアプリケーションが設計されている必要があります。アプリケーションには、できるだけ特定のマウントポイントを使用しないようにします。

SNA アプリケーションへの複数のあて先の使用

SNA は、ポイントツーポイントリンク型です。つまり、SNA サービスを別のシステムに容易に移動することはできません。これは、SNA システムが元々メインフレームで開発されたポイントツーポイントリンクという異なるリンクを使用するためです。したがって、あるノードのバックアップリンクまたは他のノードのバックアップリンクは、SNA が単一障害点にならないように構成されなければなりません。一度にアクティブにできる SNA リンクの構成は 1 つだけであることに注意してください。このため、他の目的に使用されているバックアップリンクがあれば、ミッションクリティカルな基幹業務をフェイルオーバーする場合に備えて再構成する必要があります。

ファイルロックの回避

NFS 環境では、NFS サーバー上でファイルがロックされるため、アプリケーションはファイルロック機構の使用を避けなければなりません。また、ファイルロックは、ローカルシステムとリモートシステム両方のアプリケーションで避けなければなりません。ローカルファイルのロックが使用されてシステムで障害が発生した場合、バックアップシステムとして動作するシステムは、障害が発生したシステムが持っていたロックに関する情報を持ちません。このため、アプリケーションが再起動したときに、問題が発生する場合があります。

リモートファイルロックを使用した場合は、ローカルファイルロックの場合よりも悪い状態になります。というのは、ファイルロックを行ったシステムで障害が発生する可能性があるためです。ロックを解除することは完全に不可能となり、アプリケーションの他の部分はそのデータにアクセスできなくなります。NFS 環境では、NFS クライアントシステムで障害が発生した際に、ファイルロックが原因で大きな遅れが生じ、フェイルオーバーそのものが遅れる可能性があります。

クライアント接続の復元

クライアントは、障害発生後、どのようにしてサーバーに再接続するのでしょうか。

クライアントアプリケーションを作成する際に、サーバーへの接続の切断と回復の見込みのあるアプリケーション関連のその他のエラーを明確に区別することが重要です。接続が失われた場合、アプリケーションは特別な処置を行う必要があります。

考慮する問題の 1 つは、障害発生後、新しく起動したサーバーにいつ再接続すればよいかを、クライアントがどのようにして知るかということです。通常は、クライアントは単にセッションを再起動するか、再度ログインすることだけが必要です。ただし、これは自動化された方法とはいえません。たとえば、正しく調整されたハードウェアとアプリケーションシステムが、障害発生から 5 分後にフェイルオーバーできるとします。ただしユーザーが、障害発生中に応答がないため、2 分間待って操作をあきらめてコーヒーを飲みに行き、28 分間戻らない場合は、ダウンタイムは実際には 5 分ではなく 30 分となります。したがって、再接続を試みる回

数、再接続を試みる間隔、ユーザーに接続の切断を通知するかどうか、といった要素を考慮することが必要です。

クライアントの再接続に関する設計方法は以下のとおり多数あります。

- 障害が発生したサーバーへの再接続を行うクライアントを設計する。
この作業は、ユーザーによる再接続に頼らずに、クライアントアプリケーション側の仕様として組み込んでください。サーバーがバックアップされ5分後に正常に動作し、クライアントが再接続を続行している場合は、クライアントアプリケーションは5分後にサーバーとのリンクを再度確立し、トランザクションを再開または続行します。ユーザーが介入する必要はありません。
- 別のサーバーに再接続するようにクライアントを設計する。
複数のアクティブサーバーを持つサーバーを設計している場合、クライアントは2番目のサーバーに接続するので、ユーザー側では少々遅れが発生するだけで済みます。
この設計の問題は、クライアントがいつ2番目のサーバーへ切り替えればよいかを理解しておかなければならないことです。クライアントは1番目のサーバーへの接続を中止して2番目のサーバーへ移行するまで、どれくらいの間再試行を行うのでしょうか。これには決まった答えはなく、サーバーアプリケーションの設計により決まります。障害発生後、アプリケーションを同じノード上で再起動できる場合(後述の「[アプリケーション障害の処理](#)」を参照してください)、現在のサーバーへの再試行は、サーバーをローカルで再起動するのに必要な時間だけ続行することが必要です。これにより、アプリケーション障害発生時に、クライアントが2番目のサーバーへ切り替わらないようにすることができます。
- トランザクション処理モニターまたはメッセージキューイングソフトウェアを使ってシステムを強化する。
サーバーとクライアント間のインターフェイスを提供するトランザクション処理モニター(Tuxedo または DCE/Encina など)を使用します。トランザクション処理モニター(TPM)は、ハイアベイラビリティアプリケーションの作成に有用です。トランザクションを待ち行列に入れることにより、クライアントがサーバー障害を検出しないようにすることができます。TPMの多くは、オプションで代替サーバーへの経路の自動再指定や、トランザクションの自動再試行を行えます。また、TPMを使用すれば、トランザクションを確実に終了することができます。ただし、TPMはこのような作業を行うためだけの機構ではありません。サーバーが再びオンライン状態になると、トランザクションモニターは新しいサーバーに再度接続し、トランザクションの経路指定を続行します。
- 要求のキューイング
TPMを使用する別の方法として、サーバーが使用不可能な場合に、要求を待ち行列に入れます。TPMは、サーバーが使用不可能なことをユーザーに通知するのではなく、ユーザー要求を待ち行列に入れて、サーバーが再び使用可能になったときに送信します。メッセージキューイングソフトウェアは、トランザクションだけでなくすべての種類のメッセージが確実に配信され、確認されるようにします。
メッセージのキューイングは、要求が完了したことを知らせる応答をユーザーが必要としない、または期待していない場合(つまり、アプリケーションが対話型でない場合)に限り役立つ機能です。

アプリケーション障害の処理

アプリケーションの一部または全体で障害が発生した場合、どのような状態になるでしょうか。

これまでの項では、障害を、アプリケーションの障害でなく、クラスターの他の構成要素の障害と想定して説明しました。この項では、特にアプリケーションの問題を取り上げます。たとえば、ソフトウェアのバグによりアプリケーションが異常終了したり、システムリソースの問題(スワップ/メモリ領域の不足)によりアプリケーションが使用不可能になる場合があります。この項では、このような障害から回復するためにはどのようにアプリケーションを設計すればよいかを説明します。

障害に強いアプリケーションの作成

アプリケーションは、1つの構成要素で障害が発生しても続行できるようにしなければなりません。多くのアプリケーションは、1つのノード上で複数のプロセスを実行します。あるプロセスに障害が発生した場合、他のプロセスはどうなるでしょうか。他のプロセスでも障害が発生するのでしょうか。障害が発生したプロセスは、アプリケーションの他の部分に影響を与えずに、同じノード上で再起動できるでしょうか。

1つのプロセスに障害が発生した場合、他のプロセスはその構成要素が再びオンライン状態に復帰するまで待機することが理想です。このことは、その構成要素が同じシステム上、またはリモートシステム上のどちらにある場合にも当てはまります。障害が発生した構成要素は、同じシステム上で自動的に再起動して、待機中の処理に再度加わって処理を続行できます。このタイプの障害は数秒以内に検出して再起動できるので、エンドユーザーは障害の発生をまったく認識しません。

もう1つの方法は、1つの構成要素の障害により他の構成要素をすべて停止させることです。データベース SQL サーバーで障害が発生した場合、データベースを後で復旧する必要がないように、データベースを完全に停止できることが必要です。

1つの構成要素の障害が原因でシステム全体がダウンするのは非常に悪いケースです。1つの構成要素で障害が発生して、他のすべての構成要素の再起動が必要になると、ダウンタイムが長くなるためです。

アプリケーションの監視

アプリケーションを含めて、システムの構成要素はすべて、その状態を監視する必要があります。監視方法は、表示コマンドと同じくらい簡単な場合も、SQL 照会と同じくらい複雑な場合もあります。アプリケーションが正しく動作していることを確かめる方法がなければなりません。アプリケーションで障害が発生しても、それが自動的に検出されない場合は、ユーザーがダウンタイムの原因をつきとめて、アプリケーションを回復させるのにかなりの時間がかかります。

計画的ダウンタイムの短縮

計画的ダウンタイムとは、(予期されないダウンタイムとは逆に) 予定されているダウンタイムを指します。たとえば、バックアップ、新しいリビジョンのオペレーティングシステムへのシステムのアップグレード、ハードウェアの交換などです。計画的ダウンタイムについては、アプリケーション設計者は以下の項目を考慮することが必要です。

- アプリケーションのアップグレードやパッチに必要な時間の短縮
システム管理者は、ダウンタイムを設定せずに、新しいバージョンのアプリケーションをインストールできるか。リビジョンの異なるアプリケーションは、同じシステム内で動作できるか。バージョンの異なるクライアントとサーバーは、同じシステム内で動作できるか。
- オンラインでのアプリケーションの再構成の準備
アプリケーションが使用する構成情報は、アプリケーションを停止させずに変更できるか。
- 保守作業の文書化
オペレーターは、保守作業を行う方法を知っているか。

高可用性システムについて議論する場合、予期されない障害が議論の中心となります。ただし、システムを新しいリビジョンのソフトウェアにアップグレードするのに 2 週間かかる場合、苦情が大量に発生します。

以降の項では、さまざまなタイプの計画的ダウンタイムを処理する方法を説明します。

アプリケーションのアップグレードやパッチに必要な時間の短縮

年に約一度の割合で、アプリケーションの新しいリビジョンがリリースされるとします。エンドユーザーがこの新しいリビジョンへアップグレードするのにどれくらい時間がかかるでしょうか。この答えが、ユーザーがアプリケーションをアップグレードするのに要する、計画的ダウンタイムの合計です。以下のガイドラインに従って、ダウンタイムを短縮してください。

段階的アップグレードの実行

クライアントサーバー環境で「段階的アップグレード」を行います。多数の構成要素を持つシステムの場合、よく行われるのは、システム全体を停止して、すべてのノードを新しいバージョンのソフトウェアへアップグレードしてから、すべてのノードでアプリケーションを再起動するという作業です。大規模システムでは、このシナリオではダウンしている時間が長くなってしまいます。その代替策として、段階的アップグレード機能が提供されています。段階的アップグレードでは、一度に1つの構成要素だけをアップグレードすることによって、段階的なアプローチにより新しいソフトウェアを取り入れます。たとえば、データベースサーバーを、15分のダウン時間で月曜日にアップグレードします。火曜日に、2つのノードを持つアプリケーションサーバーをアップグレードします。このとき、他のノードのアプリケーションサーバーはオンライン状態なのでダウンタイムは発生しません。水曜日に、さらに2つのアプリケーションサーバーをアップグレードします。このような方法を取ることで、同時にすべてが変更されるという問題が回避される上、長い停止期間を最小限に抑えることができます。

これに関するトレードオフは、アプリケーションソフトウェアを異なるリビジョンのソフトウェアで動作させなければならないということです。上記の例では、データベースサーバーがリビジョン5.0で動作し、一部のアプリケーションサーバーがリビジョン4.0で動作することになります。このため、このような状況に対応できるようにアプリケーションが設計されている必要があります。

リリース間でデータレイアウトを変更しない

新しいフォーマットへのデータの移行は、非常に時間のかかる作業です。また、段階的アップグレードはほとんどの場合使用できません。たとえば、データベースが1番目のノードで動作している場合、2番目のノードを新しいリビジョンのデータベースへアップグレードすることが理想的です。アップグレードが完了したとき、1番目のノードから新しくアップグレードされた2番目のノードへデータベースサーバーを移動するための短いダウンタイムのスケジュールを設定します。次にデータベースサーバーを再起動すると、1番目のノードはスタンバイ状態となっており、アップグレードの準備ができています。ただし、新しいリビジョンのデータベースが異なるデータベースレイアウトを必要とする場合、**以前の**データは、新しくアップグレードされたデータベースでは読み込めなくなります。データを新しいレイアウトへ移行するには、長いダウンタイムが必要になります。

オンラインでのアプリケーションの再構成の準備

大部分のアプリケーションには、アプリケーションが起動するときに読み込まれる数種類の構成情報があります。構成を変更するために、アプリケーションを停止して新しい構成ファイルを読み込む必要がある場合、ダウンタイムが発生します。

このダウンタイムを避けるには、アプリケーション対話型の構成ツールを使用して、オンラインの状態でも動的に変更を行います。つまり、アプリケーション対話型の構成ツールを使用することが理想的な解決策といえます。これにより、オンラインで変更が行われ、エンドユーザー側の中断はほとんどありません。このツールは、データサイズの拡張、システムへの新しいユーザーの追加、アプリケーションへの新しいユーザーの追加などの作業をすべてオンラインで実行できることが必要です。システム管理者がアプリケーションシステムに対して行わなければならない作業がすべてオンラインで実行できます。

保守作業の文書化

保守作業では、標準の手順を設定することが必要です。アプリケーション設計者は、高可用性環境と通常的环境について、保守作業を共通の手順で行えるようにしなければなりません。というのは、障害発生後にシステム全体を停止することが習慣になっているシステム管理者は、1つの障害だけを処理できるよう設計変更されたアプリケーションについても同じ手順を実行すると思われるからです。アプリケーション用のマニュアルには、代表的な保守作業について高可用性システムに関する場合の代替手順を記載しておくことが必要です。

B HA アプリケーションと Serviceguard のインテグレーション

ここでは、アプリケーションを Serviceguard 環境に統合する手順を要約して説明します。

1. このマニュアルを最後まで読む (特にクラスターとパッケージの構成に関する章と付録 A 「高可用性クラスターアプリケーションの設計」)。
2. 通常運用時のクラスターの動作を定義する。
 - 通常動作時のクラスターロックをどのように設定するか。
 - 大部分のユーザーが使用する標準構成はどれか。(ユーザー必要条件に関する情報があるか。)
 - データベースサーバーまたはアプリケーションサーバーなどの機能を個別のハードウェアシステムに分離できるか、またはすべての機能が 1 台のシステムで実行されるか。
3. フェイルオーバー運用時のクラスターの動作を定義する。
 - すべてのアプリケーションを引き継ぎノードにフェイルオーバーさせるか。
 - 複数のアプリケーションは同じノードにフェイルオーバーできるか。
 - Serviceguard が提供する機能以外に、すでにアプリケーション内に高可用性機構があるか。
4. 問題のある範囲を確認する。
 - システムのリブートまたはシステムパニックに対応するためのアプリケーションの機能はあるか。
 - 他のシステムへのフェイルオーバーの妨げとなるシステム固有の情報 (`uname()`、`gethostname()`、`SPU_ID`、または MAC アドレスなど) をアプリケーションで使用しているか。

HA アプリケーションのインテグレーション用チェックリスト

この項には、HA アプリケーションを単一システムまたは複数のシステムへ組み込むためのチェックリストを記載します。

単一システムでのアプリケーションの基本動作の定義

1. スタンドアロンシステムでのアプリケーションに対する基本動作を定義します。
 - システムの 1 つに、アプリケーション、データベース、その他必要なリソースをインストールします。このとき、Serviceguard の規則に従うようにしてください。
 - 個別の外部ボリュームグループにすべての共有データをインストールします。
 - 必要に応じてジャーナルファイルシステム (JFS) を使用します。
 - 標準テストを実施して、アプリケーションが正常に動作することを確認します。このテストは後で Serviceguard とのテストにも使用できます。可能な場合は、クライアントを通じてアプリケーションへ接続してみてください。
 - スタンドアロンシステムをクラッシュさせ、リブートした後、アプリケーションがどのようにして再起動するかテストします。以下の点に注意してください。
 - 手動で実行する手順はありますか。あれば、文書に記録してください。
 - すべての要素を `rc` スクリプトから起動できますか。
 - キーボードから入力せずにすべての要素を起動する簡単なスクリプトを作成してください。システム管理者なら何を入力するかを考えて、それをスクリプトに記述してください。

- アプリケーションを停止させる簡単なスクリプトを作成してください。システム管理者なら何を入力するかを考えて、それをスクリプトに記述してください。

HA アプリケーションの複数のシステムへの組み込み

1. 2 番目のシステムにアプリケーションをインストールします。
 - 2 番目のシステムに LVM を作成します。
 - このシステムに適切なユーザーを追加します。
 - 適切な実行可能ファイルをインストールします。
 - 1 番目のシステムで**実行されていない**アプリケーションを、2 番目のシステムで起動してみてください。このとき前述の手順で作成したスクリプトを使っても構いません。実行する際の手順に違いはありますか。アプリケーションは実行されますか。
 - 2 番目のシステムでアプリケーションが実行されるまでこの手順を繰り返します。
2. Serviceguard クラスタを構成します。
 - クラスタ構成を作成します。
 - パッケージを作成します。
 - パッケージスクリプトを作成します。
 - 前述の手順で作成した簡単なスクリプトを、パッケージ制御スクリプトのユーザー定義機能として使用します。
3. クラスタを起動して、アプリケーションが計画したとおりに実行されるどうかを確認めます。

クラスタのテスト

1. クラスタをテストします。
 - クライアントを接続します。
 - 通常のシステム負荷を与えます。
 - 1 番目のノードでパッケージを停止して、このパッケージを 2 番目のノードへ移動します。


```
# cmhaltpkg pkg1

# cmrunpkg -n node2 pkg1

# cmmodpkg -e pkg1
```
 - もう一度元のノードへ戻します。


```
# cmhaltpkg pkg1

# cmrunpkg -n node1 pkg1

# cmmodpkg -e pkg1
```
 - システムのいずれかで障害を発生させます。たとえば、ノード 1 の電源を切断します。このとき、ノード 2 でパッケージが起動することを確認してください。
 - 同様に、ノード 2 からノード 1 へのフェイルオーバーもテストします。
2. アプリケーション負荷のすべての組み合わせについてテストしてください。ユーザーの負荷が大きい場合や無負荷の場合、一括処理ジョブやオンライントランザクションなど、さまざまな状態でフェイルオーバープロセスを繰り返してください。
3. 各アプリケーション状態についてフェイルオーバーに要した時間のタイムラインを記録します。たとえば、クラスタの再構成に 45 秒、ファイルシステムでの `fsck` の実行に 15 秒、アプリケーションの起動に 30 秒、データベースの回復に 3 分、というようにタイムラインを記録します。

C 未記入のプランニングワークシート

この付録には、第4章「HA クラスターのプランニングと文書化」で説明したプランニングワークシートの未記入版を載せています。役立つと思われるワークシートをコピーして、プランニング作業の一環として記入してください。この付録には、次のワークシートがあります。

- ハードウェア用ワークシート (264 ページ)
- 電源用ワークシート (264 ページ)
- コーラムサーバー用ワークシート (265 ページ)
- ボリュームグループと物理ボリューム用ワークシート (265 ページ)
- クラスター構成用ワークシート (266 ページ)
- パッケージ構成用ワークシート (266 ページ)
- パッケージ制御スクリプト用ワークシート (従来のパッケージ) (267 ページ)

ハードウェア用ワークシート

```
=====
SPU Information:

Host Name _____ Server Series _____

Memory Capacity _____ Number of I/O Slots _____
=====
LAN Information:

Name of      Name of      Node IP      Traffic
Master _____ Interface _____ Addr _____ Type _____

Name of      Name of      Node IP      Traffic
Master _____ Interface _____ Addr _____ Type _____

Name of      Name of      Node IP      Traffic
Master _____ Interface _____ Addr _____ Type _____

=====

Quorum Server Name: _____ IP Address: _____

=====

Disk I/O Information for Shared Disks:

Bus Type _____ Slot Number _____ Address _____ Disk Device File _____

Bus Type _____ Slot Number _____ Address _____ Disk Device File _____

Bus Type _____ Slot Number _____ Address _____ Disk Device File _____

Bus Type _____ Slot Number _____ Address _____ Disk Device File _____
```

電源用ワークシート

```
=====
SPU Power:

Host Name _____ Power Supply _____

Host Name _____ Power Supply _____

=====
Disk Power:

Disk Unit _____ Power Supply _____
```

Disk Unit _____	Power Supply _____
Disk Unit _____	Power Supply _____
Disk Unit _____	Power Supply _____
Disk Unit _____	Power Supply _____
Disk Unit _____	Power Supply _____

=====

Tape Backup Power:

Tape Unit _____	Power Supply _____
Tape Unit _____	Power Supply _____

=====

Other Power:

Unit Name _____	Power Supply _____
Unit Name _____	Power Supply _____

クォーラムサーバー用ワークシート

Quorum Server Data:

=====

QS Hostname: _____ IP Address: _____

OR

Cluster Name: _____

Package Name: _____ Package IP Address: _____

Hostname Given to Package by Network Administrator: _____

=====

Quorum Services are Provided for:

Cluster Name: _____

Host Names _____

Host Names _____

Cluster Name: _____

Host Names _____

Host Names _____

Cluster Name: _____

Host Names _____

Host Names _____

ボリュームグループと物理ボリューム用ワークシート

=====

Volume Group Name: _____

Physical Volume Name: _____

Physical Volume Name: _____

Physical Volume Name: _____

=====

Volume Group Name: _____

Physical Volume Name: _____

Physical Volume Name: _____

Physical Volume Name: _____

クラスター構成用ワークシート

=====

Name and Nodes:

Cluster Name: _____

Node Names: _____

Maximum Configured Packages: _____

=====

Cluster Lock Data:

=====

If using a quorum server:

Quorum Server Host Name or IP Address: _____

Quorum Server Polling Interval: _____ microseconds

Quorum Server Timeout Extension: _____ microseconds

=====

If using a lock lun:

Lock LUN Name on Node 1: _____

Lock LUN Name on Node 2: _____

Lock LUN Name on Node 3: _____

Lock LUN Name on Node 4: _____

=====

Subnets:

Heartbeat Subnet: _____

Monitored Non-heartbeat Subnet: _____

Monitored Non-heartbeat Subnet: _____

=====

Timing Parameters:

Heartbeat Interval: _____

Node Timeout: _____

Network Polling Interval: _____

Autostart Delay: _____

=====

Access Policies

User: _____ Host: _____ Role: _____

User: _____ Host: _____ Role: _____

パッケージ構成用ワークシート

=====

Package Configuration File Data:

Package Name: _____ Package Type: _____

Primary Node: _____ First Failover Node: _____

Additional Failover Nodes: _____

Run Script Timeout: _____ Halt Script Timeout: _____

Package AutoRun Enabled? _____

Node Failfast Enabled? _____

```

Failover Policy:_____ Failback_policy:_____
Access Policies:
User:_____ From node:_____ Role:_____
User:_____ From node:_____ Role:_____
Log level_____ Log file:_____
Priority_____ Successor_halt_timeout_____
dependency_name_____ dependency_condition_____
dependency_location_____
=====
LVM Volume Groups:
vg_____vg01_____vg_____vg_____
vgchange_cmd:_____
=====
Logical Volumes and File Systems:
fs_name_____ fs_directory_____
fs_mount_opt_____ fs_umount_opt_____
fs_fsck_opt_____ fs_type_____
fs_name_____ fs_directory_____
fs_mount_opt_____ fs_umount_opt_____
fs_fsck_opt_____ fs_type_____
fs_name_____ fs_directory_____
fs_mount_opt_____ fs_umount_opt_____
fs_fsck_opt_____ fs_type_____
fs_mount_retry_count:_____
fs_umount_retry_count:_____
Concurrent mount/umount operations: _____
Concurrent fsck operations: _____
=====
Network Information:
IP_____ IP_____ IP_____ subnet_____
IP_____ IP_____ IP_____ subnet_____
Monitored subnet:_____
=====
Service Name: _____ Command: _____ Restart: _____ Fail Fast enabled: _____
Service Name: _____ Command: _____ Restart: _____ Fail Fast enabled: _____
Service Name: _____ Command: _____ Restart: _____ Fail Fast enabled: _____
=====
Package environment variable:_____
Package environment variable:_____
External pre-script:_____
External
script:_____
=====

```

パッケージ制御スクリプト用ワークシート (従来のパッケージ)

PACKAGE CONTROL SCRIPT WORKSHEET Page ____ of ____

Package Control Script Data:

PATH _____

VGCHANGE _____

VG[0] _____ LV[0] _____ FS[0] _____

VG[1] _____ LV[1] _____ FS[1] _____

VG[2] _____ LV[2] _____ FS[2] _____

FS Umount Count: _____ FS Mount Retry Count: _____

IP[0] _____ SUBNET _____

IP[1] _____ SUBNET _____

Service Name: _____ Command: _____ Restart: _____

Service Name: _____ Command: _____ Restart: _____

注記: MD、RAIDTAB、およびRAIDSTART は、非推奨となっているため使用しないでください。「ストレージのマルチパス」(75 ページ) を参照してください。

D IPv6 ネットワークのサポート

ここでは、次の、IPv6 のネットワークアドレスの特徴を具体的に説明します。

- IPv6 アドレスの種類
- ネットワーク構成の制限事項 (271 ページ)
- Linux での IPv6 の構成 (272 ページ)

IPv6 アドレスの種類

数種類のアドレス指定方式が、RFC 2373 (IPv6 のアドレス指定アーキテクチャー) で規定されています。IPv6 アドレスは、インターフェイスおよびインターフェイスセットのための 128 ビットの識別子です。RFC 2373 で定義されている IPv6 のアドレス形式には、いくつかの種類があります。IPv6 アドレスは、ユニキャスト、エニーキャスト、マルチキャストに大きく分類できます。

次の表では、IPv6 アドレスの 3 つの種類について説明します。

表 13 IPv6 アドレスの種類

ユニキャスト	単一インターフェイス用のアドレスです。ユニキャストアドレス宛てに送信されるパケットは、そのアドレスで識別されるインターフェイスに配信されます。
エニーキャスト	インターフェイスセット用のアドレスです。多くの場合、これらのインターフェイスは異なるノードに属しています。エニーキャストアドレス宛てに送信されるパケットは、そのアドレスで識別されるインターフェイスのいずれか 1 つに配信されます。エニーキャストアドレスの使用に関する標準は現在でも開発中のため、Linux では現在サポートしていません。
マルチキャスト	インターフェイスセット (通常は、異なるノードに属している) 用のアドレスです。マルチキャストアドレス宛てに送信されるパケットは、そのアドレスで識別されるすべてのインターフェイスに配信されます。

IPv4 と異なり、IPv6 にはブロードキャストアドレスがありません。これは、マルチキャストの機能がブロードキャストの機能を包含しているからです。

IPv6 アドレスのテキスト表現

IPv6 アドレスをテキスト文字列で表現する場合に、3 種類の形式があります。

- 1 番目の形式は `x:x:x:x:x:x:x:x` です。ここで、8 個の `x` はそれぞれが 16 ビット値を 16 進数で表現したもので、全体で 128 ビットのアドレスを表現します。例:
`2001:feCD:ba23:cd1f:dcb1:1010:9234:4088`
- IPv6 アドレスには、0 のビットが長く連続する場合があります。このようなアドレスをテキストで簡単に表現するために、特殊な構文が用意されています。すなわち、「`::`」を使うと、16 ビット分の 0 がいくつか存在することを意味します。「`::`」はアドレス内に 1 度だけ記述でき、アドレス内の連続する先頭や間や最後尾の 16 ビットの 0 を「`::`」で省略することができます。例:
たとえば、`fec0:1:0:0:0:0:0:1234` は、`fec0:1::1234` と表現できます。
- IPv4 ノードと IPv6 ノードが混在した環境では、IPv6 アドレスの代替形式が使われます。この形式は、`x:x:x:x:x:x:d.d.d.d` で、`x` の部分は IPv6 アドレスの上位の 96 ビットを 16 進数で表現した値であり、`d` の部分は IPv6 アドレスの下位の 32 ビットを 10 進数で表現した値です。通常、IPv4 マップド IPv6 アドレスと、IPv4 互換 IPv6 アドレスは、この形式で表現されます。これらのアドレス形式については、後述します。
例:
`0:0:0:0:0:0:10.1.2.3`
および
`::10.11.3.123`

IPv6 アドレスプレフィックス

IPv6 アドレスプレフィックスは IPv4 での CIDR に似ており、CIDR 記法を使って表現されます。IPv6 アドレスプレフィックスは、次の形式で表現されます。

IPv6-address/prefix-length: ここで、*IPv6-address* は上述したいずれかの表現の IPv6 アドレスであり、*prefix-length* はアドレスの左端の連続した何ビットがプレフィックスであるかを 10 進数で示したものです。例:

fec0:0:0:1::1234/64

この例では、アドレスの最初の 64 ビット fec0:0:0:1 がアドレスプレフィックスです。アドレスプレフィックスは、IPv6 アドレス内の何ビットがサブネットを表現しているかを示すために、IPv6 アドレスで使われます。

ユニキャスト

IPv6 ユニキャストアドレスは、いくつかの種類に分類できます。すなわち、集約可能グローバルユニキャストアドレス、サイトローカルアドレス、およびリンクローカルアドレスです。通常、ユニキャストアドレスは、論理的に次のように分割されます。

表 14

n ビット	128-n ビット
サブネットプレフィックス	インターフェイス ID

IPv6 ユニキャストアドレス内のインターフェイス ID は、リンク上のインターフェイスを識別するために使われます。インターフェイス ID は、そのリンク上で一意である必要があります。リンクは一般にサブネットプレフィックスで識別されます。

アドレスのすべてのビットが 0 のユニキャストアドレスは、未指定 (unspecified) アドレスと呼ばれます。テキスト表現では、「::」となります。

ユニキャストアドレスの ::1 つまり 0:0:0:0:0:0:0:1 は、ループバックアドレスと呼ばれます。これは、ノードがパケットを自分自身に送信するために使います。

IPv4 と IPv6 の互換性

IPv6 アドレス方式のフレームワーク内で IPv4 アドレスを使う方法がいくつかあります。

IPv4 互換 IPv6 アドレス

IPv6 へ移行する手段として、既存の IPv4 インフラストラクチャ上で IPv6 パケットをトンネリングする技術が使われます。このような方式をサポートする IPv6 ノードでは、下位の 32 ビットに IPv4 アドレスを埋め込んだ特殊な IPv6 アドレスを使います。このアドレスは IPv4 互換 IPv6 アドレスと呼ばれ、次のように表現されます。

表 15

80 ビット	16 ビット	32 ビット
ゼロ	0000	IPv4 アドレス

例:

:::192.168.0.1

IPv4 マップド IPv6 アドレス

IPv4 アドレスが埋め込まれた特殊な IPv6 アドレスがあります。このアドレスは IPv4 専用ノードのアドレスを IPv6 アドレスで表現するために使われます。特に、IPv6 と IPv4 の両方をサポートするアプリケーションでこのアドレスが使われます。このアドレスは、IPv4 マップド IPv6 アドレスと呼ばれます。このアドレスの形式は次のとおりです。

表 16

80 ビット	16 ビット	32 ビット
ゼロ	FFFF	IPv4 アドレス

例:

::ffff:192.168.0.1

集約可能グローバルユニキャストアドレス

グローバルユニキャストアドレスは、グローバルに一意的な IPv6 アドレスです。このアドレス形式は、RFC 2374 (『IPv6 の集約可能グローバルユニキャストアドレス形式』) で詳しく定義されています。形式は次のとおりです。

表 17

3	13	8	24	16	64 ビット
FP	TLA ID	RES	NLA ID	SLA ID	インターフェイス ID

各要素の意味は以下のとおりです。

FP (Format Prefix) は形式プレフィックスであり、集約可能グローバルユニキャストアドレスでは、この値は「001」です。

TLA ID (Top-Level Aggregation Identifier) は、最上位の集約 ID です。

RES (Reserved) は、将来のために予約されています。

NLA ID (Next-Level Aggregation Identifier) は、次階層の集約 ID です。

SLA ID (Site-Level Aggregation Identifier) は、サイト階層の集約 ID です。

インターフェイス ID は、インターフェイスの ID です。

リンクローカルアドレス

リンクローカルアドレスの形式は、次のとおりです。

表 18

10 ビット	54 ビット	64 ビット
1111111010	0	インターフェイス ID

リンクローカルアドレスは、単一リンク上のノードをアドレス指定するために使います。リンクローカルアドレスを送信元または送信先とするパケットは、ルーターで転送されません。

サイトローカルアドレス

サイトローカルアドレスの形式は、次のとおりです。

表 19

10 ビット	38 ビット	16 ビット	64 ビット
1111111011	0	サブネット ID	インターフェイス ID

サイトローカルアドレスはサイト内で使われます。ルーターは、サイトローカルな送信元アドレスまたは送信先アドレスを持つパケットを、サイトの外部に転送しません。

マルチキャストアドレス

マルチキャストアドレスはノードのグループに対する ID です。マルチキャストアドレスの形式は、次のとおりです。

表 20

8 ビット	4 ビット	4 ビット	112 ビット
11111111	フラグ	有効範囲	グループ ID

アドレスの先頭の「FF」は、このアドレスがマルチキャストアドレスであることを示しています。

「フラグ」フィールドは、4 つのフラグ「000T」からなります。上位の 3 ビットは予約されており、0 でなければなりません。最後のビット「T」は、恒久的に割り当てられたアドレスかどうかを示しています。0 であれば恒久的に割り当てられていることを意味し、そうでなければ一時的に割り当てられていることを意味します。

「有効範囲」フィールドは、マルチキャストグループの有効範囲を制限するために使われる 4 ビットのフィールドです。たとえば、「1」という値は、これがノードローカルなマルチキャストグループであることを示します。また、「2」という値は、有効範囲がリンクローカルであることを示します。また、「5」という値は、有効範囲がサイトローカルであることを示します。

「グループ ID」フィールドは、マルチキャストグループを識別するために使われます。よく使われるマルチキャストグループは、次のとおりです。

全ノードアドレス = FF02:0:0:0:0:0:0:1 (リンクローカル)

全ルーターアドレス = FF02:0:0:0:0:0:0:2 (リンクローカル)

全ルーターアドレス = FF05:0:0:0:0:0:0:2 (サイトローカル)

ネットワーク構成の制限事項

Serviceguard はデータおよびハートビート IP について、IPv6 をサポートしています。

Serviceguard for Linux での IPv6 のサポートに関する制限事項を以下に示します。

- 自動構成 IPv6 アドレスは、Serviceguard では *HEARTBEAT_IP* または *STATIONARY_IP* アドレスとしてはサポートされません。Serviceguard クラスター構成に含まれる IPv6 アドレスは、自動構成 (たとえば、ルーター通知による) は許されません。代わりに、*STATIONARY_IP* アドレスは、Red Hat では `/etc/sysconfig/network-scripts/ifcfg-<eth-ID>` 内で、SUSE では `/etc/sysconfig/network/ifcfg-<eth-ID>` 内で手動構成する必要があります。手順と例については、「[Linux での IPv6 の構成](#)」(272 ページ) を参照してください。
- リンクローカル IP アドレスは、パッケージ IP アドレス、*HEARTBEAT_IP* アドレス、または *STATIONARY_IP* アドレスとしてはサポートされません。必要に応じてパッケージ IP アドレスはサイトローカルまたはグローバルとすることができます。
- Serviceguard は、各ネットワークインターフェイスで、有効範囲の種類 (サイトローカルまたはグローバル) ごとに 1 つの IPv6 アドレスしかサポートしません (つまり、制限つきマルチネット)。つまり、クラスター構成ファイル内の *NETWORK_INTERFACE* には、最大で 2 つの IPv6 *HEARTBEAT_IP* アドレスまたは *STATIONARY_IP* アドレスを記述できます。1 つはサイトローカル IPv6 アドレスで、もう 1 つはグローバル IPv6 アドレスです。

注記: この制限は、**パッケージ**構成ではなく、**クラスター**構成に適用されます。この制限は、インターフェイス上でパッケージが使うことができる、同じスコープタイプ (サイトローカルまたはグローバル) の IPv6 再配置可能アドレスの数には影響しません。

- ボンディングは IPv6 アドレスでサポートされていますが、active-backup モードでのみサポートされています。
- Serviceguard が IPv6 をサポートするのは、10BT、100BT および Gigabit Ethernet などのイーサネット上だけです。

- ① **重要:** 重要な情報については、「クロスサブネット構成」(25 ページ)、クラスター構成のパラメーター (82 ページ) にあるパラメーター `HOSTNAME` `ADDRESS_FAMILY`、`QS_HOST`、`QS_ADDR` の説明、「名前解決の構成」(124 ページ)、およびお使いのバージョンに対応する Serviceguard for Linux のリリースノートも参照してください。

IPv6 アドレスを使用して Serviceguard for Linux および Quorum Server の各バージョンに接続する際の手順については、<http://www.hp.com/go/hpux-serviceguard-docs> -> [HP Serviceguard Quorum Server Software] にある最新版の『HP Serviceguard Quorum Server Version A.04.00 Release Notes』の「Configuring Serviceguard to Use the Quorum Server」を参照してください。

Linux での IPv6 の構成

Red Hat Enterprise Linux と SUSE Linux Enterprise Server には、`/sbin/ip` コマンドなど、適切な IPv6 ツールがすでにインストールされています。ここでは、システム上に IPv6 定常 IP アドレスを構成する方法について説明します。

Red Hat Linux での IPv6 の有効化

以下の行を `/etc/sysconfig/network` に追加します。

```
NETWORKING_IPV6=yes      # Enable global IPv6 initialization
IPv6FORWARDING=no        # Disable global IPv6 forwarding
IPv6_AUTOCONF=no         # Disable global IPv6 autoconfiguration
IPv6_AUTOTUNNEL=no       # Disable automatic IPv6 tunneling
```

Red Hat Linux での恒久的 IPv6 アドレスの追加

システム構成スクリプト (たとえば、`/etc/sysconfig/network-scripts/ifcfg-eth1`) を変更することで、追加できます。

```
DEVICE=eth1BOOTPROTO=static
BROADCAST=192.168.1.255
IPADDR=192.168.1.10
NETMASK=255.255.255.0
NETWORK=192.168.1.0
ONBOOT=yes
IPv6INIT=yes
IPv6ADDR=3ffe:ffff:0000:f101::10/64
IPv6ADDR_SECONDARIES=fec0:0:0:1::10/64
IPv6_MTU=1280
```

Red Hat Linux での、恒久的 IPv6 アドレスによるチャネルボンディングインターフェイスの構成

`/etc/sysconfig/network-scripts/ifcfg-bond0` 内の次のパラメーターを構成します。

```
DEVICE=bond0
IPADDR=12.12.12.12
NETMASK=255.255.255.0
NETWORK=12.12.12.0
BROADCAST=12.12.12.255
IPv6INIT=yes
IPv6ADDR=3ffe:ffff:0000:f101::10/64
IPv6ADDR_SECONDARIES=fec0:0:0:1::10/64
IPv6_MTU=1280
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

`/etc/modprobe.conf` に次の 2 行を追加し、ボンディングドライバーがリブート時にロードされるようにします。

```
alias bond0 bonding
options bond0 miimon=100 mode=1      # active-backup mode
```

SUSE での恒久的 IPv6 アドレスの追加

システム構成スクリプト (たとえば、`/etc/sysconfig/network/ifcfg-eth1`) を変更することで、追加できます。

```
BOOTPROTO=static
BROADCAST=10.10.18.255
IPADDR=10.10.18.18

MTU=""
NETMASK=255.255.255.0
NETWORK=10.10.18.0
REMOTE_IPADDR=""
STARTMODE=onboot
IPADDR1=3ffe::f101:10/64
IPADDR2=fec0:0:0:1::10/64
```

SUSE での、恒久的 IPv6 アドレスによるチャネルボンディングインターフェイスの構成

`/etc/sysconfig/network/ifcfg-bond0` 内の以下のパラメーターを構成します。

```
BOOTPROTO=static
BROADCAST=10.0.2.255
IPADDR=10.0.2.10
NETMASK=255.255.0.0
NETWORK=0.0.2.0
REMOTE_IPADDR=""
STARTMODE=onboot
IPADDR1=3ffe::f101:10/64IPADDR2=fec0:0:0:1::10/64
BONDING_MASTER=yes
BONDING_MODULE_OPTS="mode=active-backup miimon=100"
BONDING_SLAVE0=eth1BONDING_SLAVE1=eth2
```

追加する IPv6 アドレスごとに、構成ファイル内で `IPADDR <num>` を使って追加のパラメーターを指定します。

ボンディングモジュールオプションは各ボンディングデバイスファイルで指定されるため、`/etc/modprobe.conf` では何も指定する必要はありません。

E Serviceguard Manager の使用

HP Serviceguard Manager は、Web ベースの HP System Management Homepage (HP SMH) の中のツールの 1 つで、以前の Serviceguard 管理ツールの代わりとなるツールです。Serviceguard Manager では、サポートされている Web ブラウザーを使って、任意のシステムから Serviceguard クラスタを監視、管理、および構成することができます。

HP Serviceguard Manager メインページでは、各ノードとそのパッケージのステータスなど、クラスタの状態に関する概要を確認することができます。

Serviceguard Manager の最新リリース、およびインストールと構成手順については、Serviceguard for Linux のお使いのバージョンに対応するリリースノートを参照してください。

オンラインヘルプシステムについて

Serviceguard Manager を起動すると、Serviceguard Manager ツールヒントが使用できるようになります。これは、読み取り専用プロパティページで、マウスを動かしてフィールドにカーソルを合わせると、そのフィールドに対する簡単な定義を見ることができます。また、画面の右上隅にある  ボタンをクリックしてオンラインヘルプを開くこともできます。ヘルプトピック「HP Serviceguard Manager メインページについて」から始めてください。「セキュリティについて」は必ず参照してください。なぜなら、HP Serviceguard Manager のアクセス制御ポリシーとルート権限に関する説明が記載されているからです。

Serviceguard Manager の起動

ここでは、2 つの一般的なシナリオについて説明します。



ヒント: Tomcat によって報告される **メモリ不足** のエラー (Exception in thread "main" java.lang.OutOfMemoryError: Java heap space) を回避するため、以下のコマンドを実行します。このエラーは特に、サーバーに高い負荷がかかっている場合や Serviceguard Manager が大規模なクラスタ (4 ノード、300 パッケージ) を管理している場合に起こる可能性があります。

1. hpsmhd を停止します。
`/etc/init.d/hpsmhd -stop`
2. `/opt/hp/hpsmh/tomcat/bin/startup.sh` ファイルを変更して、`export` 文のセクションに次の行を追加します。
`export CATALINA_OPTS="-Xms512m -Xmx512m"`
3. ファイルを保存して、hpsmhd を再起動します。
`/etc/init.d/hpsmhd -start`

シナリオ 1 - 単一クラスタの管理

シナリオ 1 は次のような場合に適用されます。

- 単一クラスタを管理する。
- Serviceguard バージョン A.11.19 以降がインストールされている。

注記: ユーザーのクラスター管理能力は、SMH のアクセスロールに応じて次のように制限されます。

- HP SMH 管理者としてアクセスするユーザーは、クラスター管理機能をすべて利用できます。
- HP SMH オペレーター権限が付与されたユーザーは、クラスターを監視することができ、Serviceguard ロールベースアクセスで設定されている権限でクラスター管理を管理することができます。
- HP SMH ユーザーとしてアクセスするユーザーは、クラスター管理機能を利用することはできません。

1. 標準 URL `http://< サーバーの完全なホスト名 >:2301/`を入力します。
例: `http://clusternode1.cup.hp.com:2301/`
2. System Management Homepage のログイン画面が表示されたら、ログイン資格情報を入力して [Sign In] をクリックします。
選択したサーバーの System Management Homepage が表示されます。
3. [Serviceguard Cluster] ボックスでクラスターの名前をクリックします。

注記: クラスターがまだ構成されていない場合、Serviceguard クラスターセクションはこの画面に表示されません。クラスターを作成するには、まず SMH の [Tools] メニューから [Serviceguard] ボックスの [Serviceguard Manager] リンクをクリックし、次に [クラスターの作成] をクリックします。

以下の図は、HP Serviceguard Manager メインページのブラウザー画面です。

図 28 System Management Homepage と Serviceguard Manager



番号	要素	説明
1	クラスターおよび全体のステータスと警告	クラスターのステータス、警告、および全般情報が表示されます。 注記: メニュー項目 [System Tools] は、このバージョンの Serviceguard Manager では利用できません。
2	メニューツールバー	メニューツールバーは、HP Serviceguard Manager Homepage の中の、クラスター、ノード、またはパッケージの表示専用プロパティページにあります。使用可能なメニューオプションは、表示しているプロパティページの種類 (クラスター、ノード、またはパッケージ) によって異なります。
3	タブバー	タブバーには、クラスター関連の追加情報が表示されます。特定のノードまたはパッケージをクリックすると、タブバーに表示される内容が変わります。

番号	要素	説明
4	ノード情報	ノードのステータス、警告、および全般情報が表示されます。
5	パッケージ情報	パッケージのステータス、警告、および全般情報が表示されます。

シナリオ 2 - マルチクラスターの管理

複数のクラスターを管理または監視するため、新しいブラウザーセッションを開きます。
シナリオ 2 は次のような場合に適用されます。

- Serviceguard バージョン A.11.15.01～A.11.20.00 がインストールされたクラスターが 1 つ以上存在する。
- Serviceguard Manager バージョン A.05.03 と HP SIM 5.10 以降がサーバーにインストールされている。

注記: Serviceguard Manager が HP Systems Insight Manager Central Management Server にインストールされている場合は、HP Systems Insight Manager バージョン 5.10 以降であれば Serviceguard Manager を起動することができます。

Serviceguard A.11.19 クラスターの場合、Systems Insight Manager は、クラスター内のいずれかのノードから Serviceguard Manager B.02.00 を起動しようとします。Serviceguard A.11.18 クラスターの場合、Systems Insight Manager は、クラスター内のいずれかのノードから Serviceguard Manager B.01.01 を起動しようとします。

Serviceguard A.11.20.00 クラスターでは、Systems Insight Manager は、クラスター内部の 1 つのノードから Serviceguard Manager B.03.31.00 の起動を試みます。

Serviceguard A.11.16 以前のクラスターの場合は、Java Web Start を介して Serviceguard Manager A.05.01 が起動されます。各 Serviceguard ノードのホスト名が DNS によって解決されることを確認する必要があります。Serviceguard Manager のこの古いバージョンの詳細は、<http://www.hp.com/go/hpux-serviceguard-docs> -> [Serviceguard Manager] にある『Serviceguard Manager バージョン A.05.01 リリースノート』を参照してください。

1. 標準 URL `https://<SIM サーバーの完全なホスト名>:50000/` を入力します。
例: `https://SIMserver.cup.hp.com:50000/`
2. Systems Insight Manager のログイン画面が表示されたら、ログイン資格情報を入力して [サインイン] をクリックします。
3. 左側のパネルで [タイプ別クラスター] を展開します。

図 29 種類のクラスター



4. [HP Serviceguard] を展開し、該当の Serviceguard クラスターをクリックします。

注記: 以前のリリースの Serviceguard を実行しているクラスターをクリックすると、Java Webstart を介して Serviceguard Manager A.05.01 (インストールされている場合) を起動するリンクがページに表示されます。

F パラメーターの最大値と最小値

表 21 に、クラスター構成パラメーターに指定できる値の範囲を示します。

表 21 クラスター構成パラメーターの最小値と最大値

クラスターパラメーター	最小値	最大値	デフォルト値	備考
メンバータイムアウト	第 4 章の「クラスター構成パラメーター」の <i>MEMBER_TIMEOUT</i> を参照してください。	第 4 章の「クラスター構成パラメーター」の <i>MEMBER_TIMEOUT</i> を参照してください。	14,000,000 マイクロ秒	
AutoStart Timeout (自動起動タイムアウト)	60,000,000 マイクロ秒	制限なし (ULONG_MAX)	600,000,000 マイクロ秒	
Network Polling Interval (ネットワークポーリング間隔)	100,000 マイクロ秒	制限なし (ULONG_MAX)	2,000,000 マイクロ秒	
Maximum Configured Packages (構成可能最大パッケージ)	0	300	300	

ULONG_MAX の値は 4,294,967,295 であり、これが実用上の制限になります。

表 22 に、パッケージ構成パラメーターに指定できる値の範囲を示します。

表 22 パッケージ構成パラメーターの最小値と最大値

パッケージパラメーター	最小値	最大値	デフォルト値	備考
起動スクリプトのタイムアウト	10 秒	ゼロ以外の値を指定する場合は 4294 秒	0 (NO_TIMEOUT)	これは推奨値です。
停止スクリプトのタイムアウト	10 秒	ゼロ以外の値を指定する場合は 4294 秒	0 (NO_TIMEOUT)	これは推奨値です。停止スクリプトのタイムアウトの値は、サービス停止のタイムアウトのすべての値の合計よりも大きくなければなりません。
サービス停止のタイムアウト	0 秒	4294 秒	0(サービスが終了するまでの待ち時間なし)	

G 汎用リソース用の監視スクリプト

監視スクリプトは、エンドユーザーが作成するスクリプトで、リソースの監視と汎用リソースのステータス設定のためのコアロジックを含む必要があります。監視スクリプトは、パッケージの開始プロセスの中で起動されます。

- シンプルリソースや拡張リソースのステータスと値は、それぞれ `cmsetresource(1m)` コマンドを使用して設定できます。
- スクリプトで監視間隔を定義することもできます。
- 監視スクリプトの起動は Serviceguard 環境内部からでも外部からでも行えますが、内部から行う場合は監視スクリプトをサービスとして設定する必要があります。監視スクリプトは、サービスとして構成して起動することをお勧めします。

詳細は、「[監視スクリプトの起動](#)」(279 ページ) を参照してください。

スクリプトのテンプレート

監視スクリプトのテンプレートが用意されています。HP では、以下のテンプレートを提供しています。

- `generic_resource_monitor.template`

これは、`/usr/local/cmcluster/conf/examples/` ディレクトリにあります。

監視スクリプトの作成方法についてのアイデアを得るには、テンプレート(281 ページ) を参考にしてください。

リソースの監視方法はエンドユーザーに任されますが、それに合わせてスクリプトのロジックを記述する必要があります。HP では、監視スクリプトに組み込む内容は提案していませんが、次の推奨事項が役に立つことがあります。

- 監視間隔は、汎用リソースが構成されるアプリケーションパッケージによる障害検出の緊急度に基づき選択してください。
- `cmgetresource` で汎用リソースのステータスと値を取得してから、そのステータスと値を設定します。
- ステータスと値の設定は、変更されたときだけに行います。

「[シンプル/拡張汎用リソースのステータスと値の取得と設定](#)」(99 ページ)、ならびに `cmgetresource(1m)` と `cmsetresource(1m)` のマンページを参照してください。

「[汎用リソース監視サービスの使用](#)」(45 ページ) を参照してください。

監視スクリプトの起動

監視スクリプトは、次の方法で起動できます。

`evaluation_type` が `during_package_start` のリソースの場合

- 監視スクリプトはパッケージで利用できる、`service_name`、`service_cmd`、および `service_halt_timeout` というサービス関数によって起動されます。この方法では、Serviceguard による監視が行われるため、スクリプトの高可用性が確保されます。このため、この方法をお勧めします。
- 監視スクリプトは、パッケージの一部として、`external_script` または `external_pre_script` を介して起動することもできます。
- また、Serviceguard 環境外部 (`init`、`rc` スクリプトなど) で起動することもできます。(Serviceguard は、これらのスクリプトを監視しません。)
- 汎用リソースとその監視スクリプトの名前、つまり `service_name` と `generic_resource_name` を一致させる必要はありませんが、一致させるとモニターの識別が容易になるため、便利です。
- 複数のパッケージを対象に指定される共通リソースは、1 つの監視スクリプトで監視できます。

evaluation_type が before_package_start のリソースの場合

- Serviceguard 環境外部 (init、rc スクリプトなど) で起動することもできます。(Serviceguard は、これらのスクリプトを監視しません。)
- タイプが before_package_start のクラスターでは、すべてのリソースに対する監視スクリプトをサービス関数を使用して単一のマルチノードパッケージで構成でき、そのリソースを必要とする任意のパッケージがそのパッケージ構成ファイルにその汎用リソース名を記述できます。

この方法では、Serviceguard による監視が行われるため、スクリプトの高可用性が確保されます。このため、この方法をお勧めします。監視スクリプトは、パッケージが実行されるように構成されているすべてのノードで実行されるように構成する必要があります。以下の推奨事項とサンプルを参照してください。

汎用リソースのパラメーターの説明は、「[パッケージのパラメーターの説明](#)」(162 ページ)に掲載されています。

HP では以下の操作を推奨しています。

- 単一マルチノードパッケージを作成し、サービス関数を使用してそのマルチノードパッケージ内部で、タイプが before_package_start の汎用リソースについてのすべての監視スクリプトを構成します。
- アプリケーションパッケージに汎用リソース名を記述し、汎用リソースを before_package_start として構成します。
- 可読性を高めるために、アプリケーションパッケージがこのマルチノードパッケージに依存しているという依存関係を構成します。

例:

```
package_name      generic_resource_monitors
package_type      multi_node

service_name      lan1_monitor
service_cmd       $SGCONF/generic_resource_monitors/lan1.sh

service_name      cpu_monitor
service_cmd       $SGCONF/generic_resource_monitors/cpu_monitor.sh
```

上記の例は、generic_resource_monitors という名前のサンプルマルチノードパッケージを示します。また、それぞれが LAN と CPU の監視のために構成された 2 つの監視スクリプトを含みます。これらの監視スクリプトは、LAN インターフェイス、CPU を監視し、スクリプトで定義される汎用リソースのステータスを条件に合わせて設定します。

before_package_start として構成されている LAN リソースを含むパッケージ pkg1 とこのリソースの監視スクリプトが、マルチノードパッケージ generic_resource_monitors で実行されると想定します。パッケージ pkg1 を起動するためにはマルチノードパッケージが UP でなければならないという依存関係を作成します。マルチノードパッケージが起動すると、監視スクリプト 'lan1.sh' の一部としてリソース 'lan1' の監視が開始されます。このスクリプトは、汎用リソース 'lan1' のステータスを設定します。リソースが「Up」になると、パッケージ pkg1 をいつでも起動できます。

```
package_name      pkg1
package_type      failover

generic_resource_name      lan1
generic_resource_evaluation_type      before_package_start

dependency_name      generic_resource_monitors
dependency_condition      generic_resource_monitors = up
dependency_location      same_node
```

同様に、「CPU」を before_package_start として構成する必要があるもう 1 つのパッケージ pkg2 を想定します。

```
package_name      pkg2
package_type      failover
```

```

generic_resource_name      cpu
generic_resource_evaluation_type  before_package_start

generic_resource_name      lan1
generic_resource_evaluation_type  before_package_start

dependency_name            generic_resource_monitors
dependency_condition        generic_resource_monitors = up
dependency_location        same_node

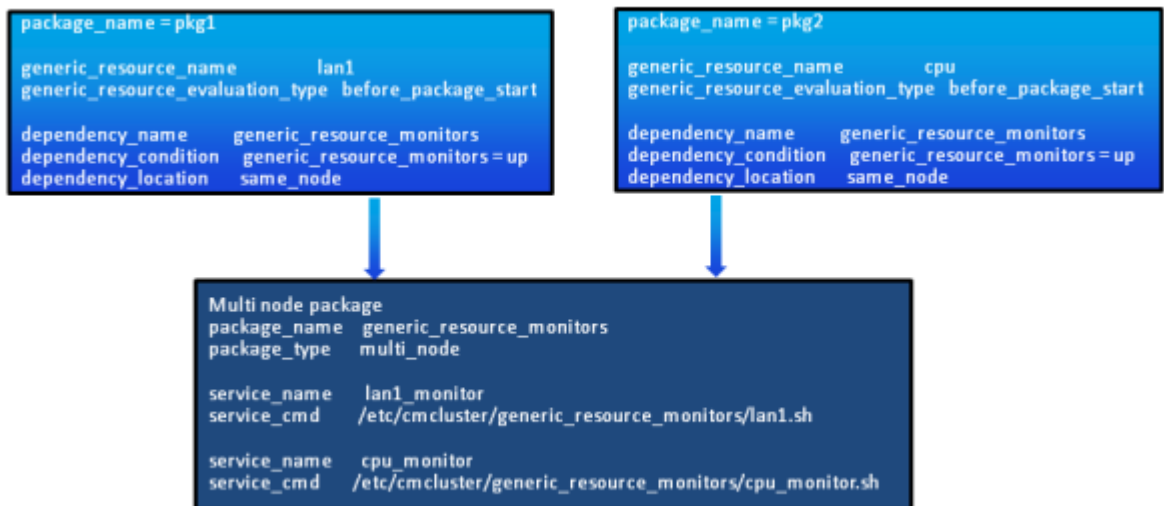
```

これにより、before_package_start タイプのすべての汎用リソースの監視スクリプトが、1つのマルチノードパッケージで構成され、この汎用リソースを必要とする任意のパッケージがこの汎用リソース名を構成できるようになります。

複数のパッケージで共通するリソースを監視する必要がある場合は、監視スクリプトを上記のマルチノードパッケージで構成でき、上の例の汎用リソース「lan1」で確認できるように、複数のパッケージがそのパッケージ構成ファイルで同じ汎用リソース名を定義できます。

図 30 に、1 つは LAN、もう 1 つは CPU の監視のために構成された 2 つの監視スクリプトを含むマルチノードパッケージを示します。2 つのパッケージは汎用リソース名で構成され、マルチノードパッケージに依存します。

図 30 before_package_start タイプの汎用リソースのすべての監視スクリプトで構成されたマルチノードパッケージ



監視スクリプトのテンプレート

監視スクリプトのテンプレート -

`/usr/local/cmcluster/conf/examples/generic_resource_monitor.template`

```

# *****
# *
# * This script is a template that can be used as a service when
# * creating a customer defined sample monitor script for
# * generic resource(s).
# *
# * Once created, this script can be configured into the package
# * configuration file as a service with the "service_name",
# * "service_cmd" and "service_halt_timeout" parameters.
# * Note that the respective "sg/service" and the
# * "sg/generic_resource" modules need to be specified in the package
# * configuration file in order to configure these parameters.
# *
# *
# * -----
# * U T I L I T Y   F U N C T I O N S
# * -----
# *

```

```

# * The following utility functions are sourced in from $SG_UTILS      *
# * ($SGCONF/scripts/mscripts/utills.sh) and available for use:      *
# *                                                                     *
# *      sg_log <log level> <log msg>                                *
# *                                                                     *
# *      By default, only log messages with a log level of 0 will    *
# *      be output to the log file. If parameter "log_level" is      *
# *      configured in the package configuration file, then log      *
# *      messages that have a log level that is equal to or          *
# *      greater than the configured log level will be output.       *
# *                                                                     *
# *      In addition, the format of the time stamp is prefixed in    *
# *      front of the log message.                                    *
# *                                                                     *
# *                                                                     *
# *****
#
#####

#####
# Initialize the variables & command paths
#####
#set the path for the command rm
<RM= PATH>
#####
# Source utility functions.
#####

if [[ -z $SG_UTILS ]]
then
    . /etc/cmcluster.conf
    SG_UTILS=$SGCONF/scripts/mscripts/utills.sh
fi

if [[ -f ${SG_UTILS} ]]
then
    . ${SG_UTILS}
    if (( $? != 0 ))
    then
        echo "ERROR: Unable to source package utility functions file: ${SG_UTILS}"
        exit 1
    fi
else
    echo "ERROR: Unable to find package utility functions file: ${SG_UTILS}"
    exit 1
fi

#####
# Source the package environment variables.
#####

typeset postfix=$(date +"%H.%M.%S")
SG_ENV_FILE=/var/tmp/${SG_PACKAGE}.${postfix}.$$tmp
$SGSBIN/cmgetpkgenv $SG_PACKAGE > $SG_ENV_FILE
if (( $? != 0 ))
then
    echo "ERROR: Unable to retrieve package attributes."
    exit 1
fi
. $SG_ENV_FILE
$RM -f $SG_ENV_FILE

#####
#
# start_command
#
# This function should define actions to take when the package starts
#
#####

function start_command
{

```

```

    sg_log 5 "start_command"

    # ADD your service start steps here

    return 0
}

#####
#
# stop_command
#
# This function should define actions to take when the package halts
#
#####

function stop_command
{
    sg_log 5 "stop_command"

    # ADD your halt steps here

    exit 1
}

#####
# main routine
#####

sg_log 5 "customer defined monitor script"

#####
#
# Customer defined monitor script should be doing following
# functionality.
#
# When the package is halting, cmhaltpkg will issue a SIGTERM signal to
# the service(s) configured in package. Use SIGTERM handler to stop
# the monitor script.
#
# Monitor the generic resource configured in package using customer
# defined tools and set the status or value to generic resource by using
# "cmsetresource" command. When setting the status or value get the current
# status or value using "cmgetresource" and set only if they are different.
#
#####

start_command $*

# SIGTERM signal handler to stop the monitor script
trap "stop_command" $SIGTERM

while [ 1 ]
do

    # Using customer defined tools get the status or value
    # of generic resource(s) which are configured in package.

    # Set the status or value of the generic resource using
    # "cmsetresource" command. Before setting the status or value
    # compare the new status or value by getting the existing status or
    # value using "cmgetresource" and set only if they are different.

    # Wait for customer defined interval to check the status or value
    # for next time.

```

H HP Serviceguard Contributed Toolkit

HP Serviceguard Contributed Toolkit Suite (Contrib toolkit) には、Linux 環境において、よく使われるアプリケーションと Serviceguard との統合を支援するツールキットがまとめられています。

このツールキットには、各自の要件に合わせてパッケージをカスタマイズする方法を記載したユーザーガイドが含まれています。詳細は、<http://www.hp.com/go/linux-serviceguard-docs> にある『HP Serviceguard Contributed Toolkit Suite on Linux Release Notes Version A.04.02.01』を参照してください。

これとは別に、NFS や Oracle のツールキットも用意されています。詳細は、<http://www.hp.com/go/linux-serviceguard-docs> を参照してください。

その他のアプリケーションインテグレーション用スクリプトをお求めの際は、当社担当者へお問い合わせください。

索引

A

ARP メッセージ
切り替え後, 63
AUTO_START
デフォルト値の影響, 69
AUTO_START_TIMEOUT
クラスター構成ファイルのパラメーター, 89
AUTO_START_TIMEOUT (自動起動遅延)
クラスターマネージャー構成のパラメーター, 89

C

CAPACITY_NAME
定義済み, 87
CAPACITY_VALUE
定義済み, 87
cmapplyconf, 213, 224
cmapplyconf コマンド, 184
cmcheckconf, 151, 184, 223
トラブルシューティング, 242
cmcheckconf コマンド, 184
cmclld デーモン
とセーフティタイマー, 30
とノード TOC, 30
とノードのリポート, 30
cmclnodelist ブートストラップファイル, 123
cmdeleteconf
クラスター構成の削除, 155
パッケージ構成の削除, 227
cmmakepkg
例, 180
cmmodnet
制御スクリプトでの IP アドレスの割り当て, 55
cmnetassist デーモン, 31
cmnetd デーモン, 29
cmquerycl
トラブルシューティング, 242
cmsnmpd デーモン, 30
CONFIGURED_IO_TIMEOUT_EXTENSION
定義済み, 90

D

DNS サービス, 126

F

FAILBACK_POLICY パラメーター
パッケージマネージャーが使用する, 43
FAILOVER_POLICY パラメーター
パッケージマネージャーが使用する, 41
FibreChannel, 27
FS, 179
サンプルのパッケージ制御スクリプト内, 221
FS_MOUNT_OPT
サンプルのパッケージ制御スクリプト内, 221

G

gethostbyname(), 256

H

HALT_SCRIPT
パッケージ構成のパラメーター, 179
HALT_SCRIPT_TIMEOUT (停止スクリプトのタイムアウト)
パッケージ構成のパラメーター, 179
HA アプリケーションと Serviceguard のインテグレーション, 262
HA クラスターのプランニングと文書化, 72
HEARTBEAT_IP
クラスター構成のパラメーター, 85
HOSTNAME_ADDRESS_FAMILY
説明と制限事項, 79
定義済み, 83

I

I/O スロット
ハードウェアのプランニング, 74, 75
I/O バスアドレス
ハードウェアのプランニング, 76
IP
サンプルのパッケージ制御スクリプト内, 221
IP_MONITOR
定義済み, 91
IP アドレス
移植可能, 55
切り替え, 40, 41, 63
ノードおよびパッケージ用, 54
ハードウェアのプランニング, 74, 78
パッケージでの追加と削除, 55
パッケージの確認, 240
IP アドレスによる負荷分散, 56
IP アドレスの切り替え, 41, 63

J

JFS, 252

L

LAN
インターフェイス名, 74, 78
ハートビート, 33
LAN インターフェイス
プライマリとセカンダリ, 23
LAN 障害
Serviceguard の動作, 23
LAN のプランニング
データ通信型, 75
ホスト IP アドレス, 74, 78
LV, 179
サンプルのパッケージ制御スクリプト内, 221
LVM
クラスターで使うコマンド, 132

ディスク, 27
プランニング, 78

M

MAC アドレス, 255
MAX_CONFIGURED_PACKAGES
 クラスターマネージャー構成のパラメーター, 92
MEMBER_TIMEOUT
 構成, 88
 最大値と最小値, 88
 定義済み, 88
 とセーフティタイマー, 30

N

NODE_FAIL_FAST_ENABLED
 設定の影響, 70
NODE_NAME
 クラスター構成のパラメーター, 84
 クラスターマネージャー構成のパラメーター, 83, 84
NTP
 クラスターのタイムプロトコル, 127

O

OTS/9000 のサポート, 278

P

PATH, 179
POLLING_TARGET
 定義済み, 91

Q

QS_ADDR
 クラスターマネージャー構成のパラメーター, 84

R

RUN_SCRIPT
 パッケージ構成のパラメーター, 179
RUN_SCRIPT_TIMEOUT (実行スクリプトのタイムアウト)
 パッケージ構成のパラメーター, 179

S

S800 シリーズ番号
 ハードウェアのプランニング, 74
SERVICE_CMD
 サンプルのパッケージ制御スクリプト内, 221
SERVICE_FAIL_FAST_ENABLED
 とノード TOC, 70
SERVICE_NAME
 サンプルのパッケージ制御スクリプト内, 221
SERVICE_RESTART
 サンプルのパッケージ制御スクリプト内, 221
Serviceguard
 インストール, 122
 概要, 18
Serviceguard Manager, 21
 概要, 20
Serviceguard コマンド
 パッケージの構成, 218
Serviceguard コマンドを使ったクラスターの監視, 152

Serviceguard でサポートされるディスク, 27
Serviceguard でサポートされるネットワーク, 23
Serviceguard について, 18
Serviceguard のインストール, 122
Serviceguard の概要, 18
Serviceguard の構成作業
 図, 22

Serviceguard のソフトウェア構成要素
 図, 29

Serviceguard の動作
 LAN 障害の場合, 23
 監視対象リソース障害の場合, 23
 ソフトウェア障害の場合, 23

Serviceguard のネットワーク構成要素の理解, 23

SMN パッケージ, 38
SNA アプリケーション, 258

SPU 情報
 プランニング, 74

STATIONARY_IP
 クラスター構成のパラメーター, 86

SUBNET
 サンプルのパッケージ制御スクリプト内, 221
 パッケージ構成のパラメーター, 179

SUBNET (IP Monitor 用)
 定義済み, 90

successor_halt_timeout パラメーター, 166

T

TOC
 セーフティタイマー, 88
 とセーフティタイマー, 30
 とパッケージの可用性, 69
 ノード障害発生時, 68

U

uname(2), 257
UPS
 電源, 28
 電源のプランニング, 76

V

VG
 サンプルのパッケージ制御スクリプト内, 221
vgcfgbackup
 ボリュームグループ構成のバックアップに使用, 140
VGChange, 179
VGCHANGE
 パッケージ制御スクリプト内, 221

W

WEIGHT_DEFAULT
 定義済み, 92
WEIGHT_NAME
 定義済み, 92

あ

アクセス制御ポリシー, 146
アクティブノード, 19
アプリケーション

- Serviceguard との統合手順のチェックリスト, 262
- 自動化, 250
- 障害の処理, 259
- ネットワークの HA サービスの記述, 251
- アプリケーション障害の処理, 259
- アプリケーション操作の自動化, 250
- アプリケーションでのクライアント接続の復元, 258
- アプリケーションのフェイルオーバー速度の制御, 251

い

- イーサネット
 - 冗長構成, 24
- 依存関係
 - 構成, 100
- 移動性 IP アドレス
 - Serviceguard パッケージ内, 55
 - 定義済み, 54

お

- 応答
 - パッケージ障害とサービス障害, 70
- オフラインのクラスターでのパッケージの再構成, 227

か

- カーネル
 - セーフティタイマー, 30
 - ハング、および TOC, 68
- カーネルの一貫性
 - クラスター構成の, 127
- カーネル割り込み
 - と TOC の可能性, 88
- 回復時間, 79
- 概要
 - Serviceguard ソフトウェアの理解, 29
 - Serviceguard ハードウェアの理解, 23
 - Serviceguard の概要, 18
- 拡張
 - ブランニング, 96
- 稼働中のクラスターでのパッケージの削除, 227
- 稼働中のクラスターからのノードの削除, 214
- 稼働中のクラスターからのパッケージの削除, 184
- 稼働中のクラスターの運用からのノードの削除, 198
- 稼働中のクラスターの再構成, 213
- 稼働中のクラスターへのノードの追加, 197
- 監視される非ハートビートサブネット
 - クラスター構成のパラメーター, 86
- 監視スクリプト
 - 起動, 279
 - テンプレート、サンプル, 281
- 監視スクリプトの起動, 279
- 監視対象リソース障害
 - Serviceguard の動作, 23
- 管理
 - オフラインのクラスターでのパッケージの再構成, 227
 - 稼働中のクラスターでのパッケージ再構成, 226
 - 稼働中のクラスターの運用からのノードの削除, 198
 - 稼働中のクラスターへのノードの追加, 197
 - クラスター, 196
 - クラスターイベント発生時の対処, 232

- クラスター全体の停止, 198
- クラスターの再構成, 212
- 構成ファイルの確認, 242
- トラブルシューティング, 240
- パッケージとサービスの, 203
- パッケージの移動, 204
- パッケージの起動, 203
- パッケージの停止, 203

き

- 起動とシャットダウン
 - アプリケーションの場合の定義, 251
- 共有ディスク
 - ブランニング, 75
- 切り替え
 - 切り替え後の ARP メッセージ, 63
 - リモートシステムの切り替え, 62
 - ローカルインターフェイスの切り替え, 56
- 切り替え IP アドレス, 40

く

- フォーラム
 - クラスターの再編成, 68
- フォーラムサーバー
 - インストール, 132
 - クラスターマネージャー構成のパラメーター, 84
 - とセーフティタイマー, 30
 - ブランニング, 78
- クラスターの停止, 198
- クライアント接続
 - アプリケーションでの復元, 258
- クラスター
 - Serviceguard, 18
 - 構成要素の冗長性, 23
 - 構成要素の理解, 23
 - コマンドで構成する, 141
 - 代表的な構成, 18
- クラスター拡張
 - 事前のブランニング, 72
- クラスター拡張のブランニング, 72
- クラスター管理, 196
 - 問題の解決, 243
- クラスター構成
 - クラスター構成の確認, 151
 - クラスターを認識するボリュームグループの識別, 145
 - すべてのノード上のファイル, 33
 - ブランニング, 79
 - ブランニングワークシート, 92
- クラスター構成とパッケージ構成の確認, 223
- クラスター構成とパッケージ構成の検証, 184
- クラスター構成とパッケージ構成の配布, 184, 223
- クラスター構成の確認, 151
- クラスター構成の削除
 - cmddeleteconf の使用, 155
- クラスター構成ファイル
 - 自動起動遅延パラメーター (AUTO_START_TIMEOUT), 89
- クラスターコーディネーター
 - 定義済み, 33

- クラスター全体の再構成, 212
- クラスター全体の停止, 198
- クラスターとノードの管理, 196
- クラスターとパッケージの管理, 186
- クラスターの運用テスト, 234
- クラスターノード
 - クラスターマネージャー構成のパラメーター, 83, 84
- クラスターノードの追加
 - 拡張計画, 120
- クラスターの稼働中にクラスターへの変更が可能, 213
- クラスターの稼働中に許されるパッケージ変更, 228
- クラスターの起動
 - 手動, 34
- クラスターの構築
 - ウォールサーバーの指定, 143
 - クラスター構成の確認, 151
 - ハートビートサブネットの識別, 145
 - 論理ボリュームインフラストラクチャ, 132
- クラスターのサイズ
 - 変更への対応, 120
- クラスターの再編成
 - シナリオ, 68
- クラスターの実行
 - パッケージの追加と削除, 184
- クラスターの自動起動
 - セットアップ, 153
- クラスターの自動再起動, 34, 198
- クラスターのトラブルシューティング, 234
- クラスターパラメーター
 - 初期構成, 33
- クラスターマネージャー
 - defined, 33
 - 主な機能, 33, 54
 - 監視される非ハートビートサブネット, 86
 - ウォールサーバーのパラメーター, 84
 - クラスターノードのパラメーター, 83, 84
 - クラスターの自動再起動, 34
 - クラスターの初期構成, 33
 - 構成できるパッケージの最大数, 92
 - 構成のプランニング, 82
 - テスト, 235
 - 動的再編成, 34
 - ネットワークポーリング間隔パラメーター, 89, 92
 - ハートビートサブネットパラメーター, 85
 - 未記入のプランニングワークシート, 266
 - メンバータイムアウトパラメーター, 88
- クラスターマネージャーの動作, 33, 54
- クラスターマネージャーのパラメーター
 - 初期構成, 33
- クラスターメンバーの変更, 34
- クラスターロック
 - 2つのノード, 35, 36
 - 4つ以上のノード, 36, 37
 - クラスターの再編成、例, 69
 - クラスターを再編成する上での使用, 35, 36
 - 構成ファイルでの特定, 143
 - と電源, 28
 - ロックなし, 37
- クラスターロックなし

- 選択, 37
- クラスターロックの使用, 35, 36
- クラスターを認識するボリュームグループの識別, 145

け

- 計画的ダウンタイムの短縮, 260

こ

- 高可用性向けに接続されたミラー化ディスク
図, 28
- 交代用スタンバイ
 - パッケージポリシーの設定, 42
 - フェイルオーバーポリシーの設定, 42
- 構成
 - 基本的な作業と手順, 22
 - クラスターの, 33
 - クラスターのプランニング, 79
 - サービス, 157
 - パッケージ, 157
 - パッケージプランニング, 92
- 構成ファイル
 - クラスターマネージャーの, 33
 - トラブルシューティング, 242
- 故障停止
 - ユーザーが影響を受けないようにする, 251

さ

- サービス障害
 - 応答, 70
- サービスの管理, 203
- サービスの構成
 - 手順, 157
- サービスの再起動, 71
- 再開可能なトランザクション, 253
- 再起動
 - クラスターの自動再起動, 34
 - 障害発生後, 71
- 再配置可能 IP アドレス
 - Serviceguard パッケージ内, 55
 - 定義済み, 54
- 再編成
 - クラスターの, 34
- サブネット
 - ハードウェアのプランニング, 74, 78
 - パッケージ構成のパラメーター, 179

し

- システム上の Serviceguard の削除, 233
- システムマルチノードパッケージ, 38, 158
- システムメッセージ
 - クラスター用の変更, 154
- システムログ, 236
- システムログファイル
 - トラブルシューティング, 241
- 自動起動遅延
 - クラスター構成ファイルのパラメーター, 89
- 自動的なクラスターの再起動, 198
- 自動フェイルバック
 - フェイルオーバーポリシーの設定, 43

- シャットダウンと起動
 - アプリケーションの場合の定義, 251
- 手動によるクラスタの起動, 34
- 障害
 - アプリケーション, 259
 - 応答の種類, 68
 - 障害後のサービスの再起動, 71
 - ネットワーク通信, 71
 - ハードウェア障害への応答, 69
 - パッケージ障害とサービス障害への応答, 70
- 障害箇所
 - ネットワーク, 24
- 障害への応答, 68
- 冗長 LAN
 - 図, 25
- 冗長イーサネット構成, 24
- 冗長性
 - クラスタ構成要素の, 23
 - ネットワーク, 24
- 冗長ネットワーク
 - ハートビート, 19

す

- 図
 - Serviceguard クラスタの構成作業, 22
 - Serviceguard のソフトウェア構成要素, 29
 - 高可用性向けに接続されたミラー化ディスク, 28
 - 冗長 LAN, 25
 - 代表的なクラスタ構成, 19
 - フェイルオーバー後の代表的なクラスタ, 20
- ステータス
 - cmviewcl, 186
 - システムログファイル, 241
 - パッケージ IP アドレス, 240

せ

- 制御スクリプト
 - 追加製品のサポート, 223
 - トラブルシューティング, 242
 - パッケージ構成内, 221
 - パッケージ構成のパス名パラメーター, 179
 - ユーザー定義関数の追加, 222
- セーフティタイマー
 - 持続時間, 30
 - と syslog, 30
 - とノード TOC, 30
- セカンダリ LAN インターフェイス
 - 定義, 23
- 説明
 - パッケージパラメーター, 162

そ

- 高可用性, 18
 - HA クラスタ, 23
 - プランニングの目的, 72
- ソフトウェア障害
 - Serviceguard の動作, 23
- ソフトウェアのインストール
 - フォーラムサーバー, 132

- ソフトウェアのプランニング
 - IVM, 78

た

- 対処
 - クラスタイベントへの, 232
- 代表的なクラスタ構成
 - 図, 19
- タイムプロトコル (NTP)
 - クラスタ, 127
- ダウンタイム
 - 計画の短縮, 260
- 単一障害点
 - 回避, 18
- 単一ノード運用, 154, 232

ち

- チェックポイント, 253

て

- 定常 IP アドレス, 54
- ディスク
 - Serviceguard でサポートされる種類, 27
 - Serviceguard では, 27
 - インターフェイス, 27
 - 交換, 236
 - 構成例, 28
 - データ, 27
 - ルート, 27
- ディスク監視
 - 構成, 184
- ディスク構成例, 28
- ディスク入出力
 - ハードウェアのプランニング, 75
- ディスクのエンクロージャー
 - 障害が発生したドライブの交換, 236
- ディスクの監視, 184
- ディスクの交換, 236
- ディスクレイアウト
 - プランニング, 78
- ディスク論理装置
 - ハードウェアのプランニング, 76
- データ
 - ディスク, 27
- データ通信型
 - LAN のハードウェアのプランニング, 75
- データ輻輳, 33
- テスト
 - クラスタマネージャー, 235
 - パッケージマネージャー, 234
- 電源
 - UPS, 28
 - とクラスタロック, 28
 - 未記入のプランニングワークシート, 264
- 電源のプランニング
 - 電源, 76
 - ワークシート, 77, 265

と

稼働中のクラスターへのパッケージの追加, 184, 227
動的クラスター再編成, 34
トラブルシューティング
 cmquerycl と cmcheckconf の使用, 242
 アプローチ, 240
 システムログファイルの確認, 241
 制御スクリプトの確認, 242
 ディスクの交換, 236
 ハードウェアの監視, 236
 パッケージ IP アドレスの確認, 240

な

名前解決サービス, 126

ね

ネットワーキング
 冗長なサブネット, 74
ネットワーク
 HA サービスとしてのネットワークアプリケーション
 の記述, 251
 IP アドレスと名前付け, 255
 IP アドレスによる負荷分散, 56
 IP アドレスへのバインディング, 257
 IP アドレスを使用するパッケージ, 256
 OTS/9000 のサポート, 278
 Serviceguard でサポートされる種類, 23
 冗長性, 24
 ノードおよびパッケージの IP アドレス, 54
 パッケージ IP アドレスの追加と削除, 55
 ポートアドレスへのバインディング, 257
 リモートシステムの切り替え, 62
 ローカルインターフェイスの切り替え, 56
ネットワークインターフェイスの冗長性, 23
ネットワーク構成要素
 Serviceguard, 23
ネットワークタイムプロトコル (NTP)
 クラスター, 127
ネットワーク通信障害, 71
ネットワークのプランニング
 サブネット, 74, 78
ネットワークポーリング間隔
 (NETWORK_POLLING_INTERVAL)
 クラスターマネージャー構成のパラメーター, 89, 92
ネットワークマネージャー
 パッケージ IP アドレスの追加と削除, 55

の

ノード
 IP アドレス, 54
 Serviceguard クラスター内, 18
 基本概念, 23
 タイムアウトと TOC の例, 68
 停止 (TOC), 68
ノードタイプ
 アクティブ, 19
 プライマリ, 19
ノードの最大数, 23

は

ハードウェア
 監視, 236
 電源, 28
ハードウェア障害
 応答, 69
ハードウェア障害への応答, 69
ハードウェアの監視, 236
ハードウェアのプランニング
 I/O スロットの数, 74
 I/O スロット番号, 75
 I/O バスアドレス, 76
 LAN インターフェイス名, 74, 78
 LAN データ通信の型, 75
 S800 シリーズ番号, 74
 SPU 情報, 74
 共有ディスクのディスク入出力情報, 75
 構成のプランニング, 74
 サブネット, 74, 78
 ディスク入出力のバス形式, 75
 ホスト IP アドレス, 74, 78
 ホスト名, 74
 未記入のプランニングワークシート, 264
 メモリ容量, 74
 ワークシート, 76
ハートビートサブネットアドレス
 クラスター構成のパラメーター, 85
ハートビートメッセージ
 定義済み, 33
ハードビートメッセージ, 19
排他的アクセス
 TOC による解放, 69
バインディング
 ネットワークアプリケーションで, 257
バス形式
 ハードウェアのプランニング, 75
パッケージ
 cmclld による管理, 30
 Serviceguard クラスター, 18
 移動, 204
 オフラインのクラスターでの再構成, 227
 起動, 203
 基本概念, 23
 クラスター稼働中の再構成, 226
 クラスターの稼働中に許される変更, 228
 実行する時期と場所の決定, 39
 タイプ, 158
 停止, 203
 パッケージ IP アドレスの追加と削除, 55
 パラメーター, 162
 パラメーターの説明, 162
 未記入のプランニングワークシート, 266, 267
 リモート切り替え, 62
 ローカルインターフェイスの切り替え, 56
パッケージ IP アドレス
 確認, 240
 定義, 55
 定義済み, 54
パッケージ管理, 203

- 問題の解決, 243
- パッケージ切り替え動作
 - 変更, 205
- パッケージ構成
 - 実行スクリプトと停止スクリプトのタイムアウトパラメーター, 179
 - プランニング, 92
- パッケージ構成の削除
 - cmddeleteconf の使用, 227
- パッケージ構成の作成, 218
- パッケージ構成パラメーター, 162
- パッケージ構成ファイル, 162
 - successor_halt_timeout, 166
 - 生成, 179
 - パッケージの依存関係のパラメーター, 167
 - 編集, 181
- パッケージコーディネーター
 - 定義, 33
- パッケージ障害
 - 応答, 70
- パッケージ制御スクリプト
 - FS パラメーター, 179
 - LV パラメーター, 179
- パッケージ制御スクリプト内のファイルシステム名パラメーター, 179
- パッケージ制御スクリプト内の論理ボリュームパラメーター, 179
- パッケージとクラスターの管理, 186
- パッケージとサービスの構成, 157
- パッケージの依存関係
 - successor_halt_timeout, 166
 - パラメーター, 167
- パッケージの移動, 204
- パッケージの起動, 203
- パッケージの構成
 - Serviceguard コマンドの使用, 218
 - 検証, 184
 - 構成の確認, 223
 - 構成の検証, 184
 - 構成ファイルの配布, 184, 223
 - サブネットのパラメーター, 179
 - 適用, 184
 - 手順, 157
 - パッケージ制御スクリプトの作成, 221
- パッケージの再構成
 - クラスターの稼働中, 226
- パッケージの停止, 203
- パッケージのファイルオーバー動作, 96
- パッケージマネージャー
 - テスト, 234
 - 未記入のプランニングワークシート, 266, 267
- パッケージモジュール, 159
 - オプション, 160
 - ベース, 159
- パッケージを実行する時期と場所の決定, 39
- パラメーター
 - パッケージの構成, 162
 - フェイルオーバー用, 96
- 汎用リソース

- 監視スクリプト, 279
- サンプル監視スクリプト, 281
- パッケージリソースの監視, 45
- 汎用リソース監視サービス
 - 使用, 45
- 汎用リソース用の監視スクリプト, 279
- 汎用リソース用のサンプル監視スクリプト, 281

ひ

- 引き継ぎノード, 19

ふ

- ファイルオーバー動作
 - パッケージ, 96
- ファイルオーバーポリシー
 - パッケージマネージャーが使用する, 41
- ファイルシステム
 - プランニング, 78
- ファイルロック, 258
- フェイルオーバー
 - アプリケーションの速度の制御, 251
 - 定義済み, 19
- フェイルオーバー後の代表的なクラスター図, 20
- フェイルオーバーパッケージ, 38, 158
- フェイルバックポリシー
 - パッケージマネージャーが使用する, 43
- 複数のシステム
 - アプリケーションの設計, 254
- 複数のシステムで実行されるアプリケーションの設計, 254
- 物理ボリューム
 - クラスターロック用, 35, 36
 - プランニング, 78
 - 未記入のプランニングワークシート, 265
- プライマリ LAN インターフェイス
 - 定義, 23
- プライマリノード, 19
- プランニング
 - SPU 情報, 74
 - 概要, 72
 - 拡張, 96
 - クォーラムサーバー, 78
 - クラスター構成, 79
 - クラスターマネージャーの構成, 82
 - クラスターロックとクラスター拡張, 77
 - 高可用性の目的, 72
 - ディスク入出力情報, 75
 - 電力, 76
 - ハードウェア構成, 74
 - パッケージ構成, 92
 - ボリュームグループと物理ボリューム, 78
 - ワークシート, 76
- プランニング全般, 72
- プランニングワークシート
 - 未記入, 264
- ブリッジネットワーク
 - 定義, 23
- ブロードキャストストーム

と TOC の可能性, 88

ほ

ポート

集結されたデュアルおよびシングルポート, 57

ホスト IP アドレス

ハードウェアのプランニング, 74, 78

ホスト名

ハードウェアのプランニング, 74

ボリュームグループ

クラスターロック用, 35, 36

プランニング, 78

ボリュームグループと物理ボリュームのプランニング,
78

ま

マルチノードパッケージ, 38, 158

め

メモリ要件

Serviceguard のロック可能メモリ, 72

メモリ容量

ハードウェアのプランニング, 74

メンバー変更

理由, 34

も

問題の解決, 243

ゆ

ユーザー定義関数

制御スクリプトへの追加, 222

り

リソース

ディスク, 27

リモート切り替え, 62

リンクレベルのアドレス, 255

ろ

ローカル切り替え, 56

ロック

クラスターロックディスクの使用, 35

クラスターロックと電源, 28

クラスターロックの使用, 36

ロックボリュームグループ、再構成, 212

ロックボリュームグループの再構成, 212

論理ボリューム

インフラストラクチャの作成, 132

プランニング, 78

わ

ワークシート

クラスター構成, 92, 266

電源構成, 77, 264, 265

ハードウェア構成, 76, 264

パッケージの構成, 266, 267

プランニングでの使用, 72

未記入, 264